Chair of Metal Forming

Master's Thesis

# Implementation of automated, interconnective Finite Element Analyses for the development of Cyber Physical Production Systems

Corinna Waiguny, BSc

September 2022

# Acknowledgement

First and foremost, I would like to express my special thanks to my supervisors, Dipl.-Ing. Marcel Sorger, who always supported and guided me, and Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Stockinger, the head of the Chair of Metal Forming. I would also like to show gratitude to Dipl.-Ing. Dr.mont. Benjamin Ralph for the discussions and his expertise. Further, I would like to thank all the Department faculty members who were involved in the process. Most importantly, I want to thank my boyfriend, my family, and my friends for the help and encouragement.

# Abstract

In recent years, Industry 4.0 - with the aim to combine production processes with state-of-the-art communication and information technologies - has led to significant changes in the industrial environment. Due to the appearance of new challenges, companies need to adapt to upcoming demands, by implementing Industry 4.0 enabling technologies, such as simulations and innovative modelling approaches. Thereby, Simulation and Modelling refers to the application of models, representing a product, system, or process, to predict model behavior and further, to extend knowledge of the model. In the metal forming industry, simulations show great potential in the design and optimization of forming processes. Through the targeted use, expensive, and time-consuming experiments can be reduced. Furthermore, the process of decision making is supported and the efficiency of forming processes can be increased.

During this thesis, models are developed to reproduce the entire upsetting process, starting at the heating of the cylindrical specimen to the transport and to the upsetting in the hydraulic press. Subsequently, an automated simulation sequence is implemented by using Python, which enables to create, run, and evaluate simulations with variable input parameters. For the calibration and validation of the developed simulations, upsetting tests with cylindrical specimen from aluminum alloy EN AW-6082 were conducted. Thereby, experiments, differing in process settings, such as temperature, transfer time, specimen geometry and upset height, were performed. The furnace and the hydraulic press at the Chair of Metal Forming represent two Cyber Physical Production Systems (CPPSs), providing sensor data of the conducted experiments. Furter, a concept is introduced, to visualize and process the sensor data to directly compare experiments and simulation.

# Kurzfassung

In den vergangenen Jahren hat die Industrie 4.0 – mit dem Ziel Produktionsprozesse mit modernster Kommunikations- und Informationstechnologie zu verbinden - zu signifikanten Veränderungen im industriellen Bereich geführt. Durch das Auftreten von neuen Problemstellungen, müssen sich Firmen an die künftigen Anforderungen anpassen und Kerntechnologien der Industrie 4.0, wie beispielsweise Simulationen und innovative Modellierungsansätze, implementieren. Simulation und Modellierung steht hierbei für die Anwendung von Modellen, welche Produkte, Systeme oder Prozesse repräsentieren, um Vorhersagen über das Modellverhalten zu treffen und zusätzlich das Wissen über das Modell zu erweitern. In der Metallumformung zeigen Simulationen ein großes Potential im Design und der Optimierung von Umformprozessen. Durch den gezielten Einsatz können kostenintensive und zeitaufwändige Experimente reduziert werden. Außerdem kann der Prozess der Entscheidungsfindung unterstützt sowie die Effizienz der Umformprozesse gesteigert werden.

Im Zuge der Arbeit wurden Simulationsmodelle erstellt, um den gesamten Prozessablauf eines Stauchversuches, beginnend beim Vorwärmen der Zylinderprobe im Ofen, über den Transport bis hin zum Stauchen mit der hydraulischen Presse, nachzubilden. Anschließend wurde mittels Python eine automatisierte Simulationsabfolge realisiert, welche es ermöglicht, Simulationen mit variablen Eingabeparametern, zu erstellen, auszuführen und auszuwerten. Für die Kalibrierung und Validierung wurden Stauchversuche von Zylinderproben aus der Aluminiumlegierung EN AW-6082 durchgeführt. Die Experimente unterschieden sich dabei in den Prozesseinstellungen hinsichtlich Temperatur, Transferzeit, Probengeometrie und Stauchhöhe. Der industrielle Ofen und die hydraulische Presse am Lehrstuhl für Umformtechnik stellen zwei Cyber Physical Production Sytems (CPPSs) dar, welche die Sensordaten der durchgeführten Versuche zur Verfügung stellen. Zudem wird ein Konzept vorgestellt, um die von den CPPSs gelieferten Sensordaten zu visualisieren und weiteres automatisch zu verarbeiten, um Experiment und Simulation direkt miteinander zu vergleichen.

# Table of Contents

1. Introduction ..... 1
2. State of the art ..... 3
3. Fundamentals ..... 4
   3.1. Forming technology ..... 4
      3.1.1. Flow stress ..... 6
      3.1.2. Cold forming and hot forming ..... 7
      3.1.3. Upsetting ..... 7
      3.1.4. Friction ..... 8
      3.1.5. Thermal effects during forming ..... 10
      3.1.6. Process parameters ..... 10
   3.2. Aluminum alloys ..... 11
   3.3. Heat transfer ..... 12
      3.3.1. Conduction ..... 12
      3.3.2. Convection ..... 12
      3.3.3. Radiation ..... 13
      3.3.4. Transient heat transfer ..... 14
   3.4. Industry 4.0 ..... 15
   3.5. Finite Element Method (FEM) ..... 17
   3.6. Metal forming simulations ..... 18
   3.7. Material models ..... 19
4. Experimental setup ..... 21
   4.1. Furnace ..... 21
   4.2. Hydraulic Press ..... 22
   4.3. Procedure ..... 23
      4.3.1. Experiment 1 ..... 24
      4.3.2. Experiment 2 ..... 25
   4.4. Sensor data visualization ..... 25
   4.5. Data processing ..... 31
5. Process simulation ..... 32
   5.1. Process cycle ..... 32
   5.2. Unit system ..... 32
   5.3. Simulation 1 - Heating ..... 33
   5.4. Simulation 2 - Transport ..... 34
   5.5. Simulation 3 – Rest on die ..... 35

5.6. Simulation 4 - Upsetting ... 35
5.7. FE mesh ... 37
5.8. Simulation outputs ... 38
5.9. Material properties ... 38
5.10. Temperature increase associated with plastic deformations ... 42
5.11. Thermal contact conductance ... 42
5.12. Convection coefficient ... 43
6. Abaqus Scripting ... 44
6.1. Abaqus Scripting Interface ... 44
6.2. Recording Python commands ... 45
6.3. Object-oriented programming ... 45
6.4. Abaqus Output Database (ODB) ... 45
6.5. Script structure ... 46
7. Concept and Implementation ... 49
7.1. Overall process ... 49
7.2. Automation of the simulation sequence ... 50
8. Evaluation and Results ... 54
8.1. Interpretation of the sensor data ... 54
8.2. Influences on the simulations ... 61
8.2.1. Heating ... 61
8.2.2. Transport ... 63
8.2.3. Rest on die ... 65
8.2.4. Upsetting ... 66
8.3. Challenges with automated models ... 70
8.4. Comparison between experiment and simulation ... 71
9. Conclusion and outlook ... 74
List of Figures ... 76
List of Tables ... 78
Appendix ... 83
Appendix A: measurements.py ... 83
Appendix B: main_script.py ... 91
Appendix C: compare.py ... 94
Appendix D: simulation_1.py ... 104
Appendix E: simulation_2p.py ... 108
Appendix F: simulation_3p.py ... 110
Appendix G: simulation_4i.py ... 113

Appendix H: simulation_4e.py ..................................................................................................... 116
Appendix I: odb_data.py ............................................................................................................ 119
Appendix J: material_data.py ....................................................................................................... 122
Appendix K: abaqus_functions.py ................................................................................................ 125
Appendix L: Documentation of experiments ................................................................................. 139

# List of Symbols

## Upper Case

| Symbol | Unit | Description |
|---|---|---|
| $A$ | [N/m$^2$] | Johnson-Cook parameter: Yield strength |
| $A_c$ | [m$^2$] | Cross sectional area |
| $A_S$ | [m$^2$] | Surface area |
| $A_{s,h}$ | [m$^2$] | Portion of the surface area |
| $B$ | [N/m$^2$] | Johnson-Cook parameter: Strain hardening constant |
| $B_i$ | [-] | Biot number |
| $C$ | [-] | Johnson-Cook parameter: Strain rate sensitivity |
| $D_1$ | [-] | Johnson-Cook Damage parameter 1 |
| $D_2$ | [-] | Johnson-Cook Damage parameter 2 |
| $D_3$ | [-] | Johnson-Cook Damage parameter 3 |
| $D_4$ | [-] | Johnson-Cook Damage parameter 4 |
| $D_5$ | [-] | Johnson-Cook Damage parameter 5 |
| $E$ | [N/m$^2$] | Young`s modulus |
| $F$ | [N] | Force |
| $F_{max}$ | [N] | Maximum force |
| $F_N$ | [N] | Normal acting force |
| $F_R$ | [N] | Frictional force |
| $L$ | [m] | Length |
| $T$ | [°C] | Temperature |
| $T_a$ | [°C] | Ambient temperature |
| $T_F$ | [°C] | Furnace temperature |
| $T_i$ | [°C] | Temperature at the time increment i |
| $T_{i+1}$ | [°C] | Temperature at the time increment i+1 |
| $T_m$ | [°C] | Melting temperature |
| $T_R$ | [°C] | Recrystallization temperature |
| $T_S$ | [°C] | Surface temperature |
| $T_t$ | [°C] | Reference temperature |
| $T_0$ | [°C] | Initial temperature |
| $T_1$ | [°C] | Temperature at position 1 |
| $T_2$ | [°C] | Temperature at position 2 |
| $T1_{end}$ | [°C] | Temperature at the end of simulation 1 |

| | | |
|---|---|---|
| $T2_{end}$ | [°C] | Temperature at the end of simulation 2 |
| $T3_{end}$ | [°C] | Temperature at the end of simulation 3 |
| $T_{\infty}$ | [°C] | Temperature of the fluid |
| $\Delta T_D$ | [°C] | Temperature difference due to dissipated deformation energy |
| $\Delta T_F$ | [°C] | Temperature difference due to friction |
| $\Delta T_T$ | [°C] | Temperature difference due to cooling of cooler dies |
| $V$ | [m$^3$] | Volume |
| $V_0$ | [m$^3$] | Initial volume |
| $V_1$ | [m$^3$] | Volume after forming |

## Lower Case

| **Symbol** | **Unit** | **Description** |
|---|---|---|
| $b_0$ | [m] | Initial width |
| $b_1$ | [m] | Width after forming |
| $c$ | [J/kg°C] | Specific heat capacity |
| $c_0$ | [m/s] | Speed of sound |
| $d$ | [m] | Diameter |
| $d_0$ | [m] | Initial diameter of the specimen |
| $d_1$ | [m] | Diameter of the specimen after forming |
| $h$ | [m] | Height |
| $h_c$ | [W/m$^2$°C] | Heat transfer coefficient for convection |
| $h_i$ | [m] | Initial distance between dies |
| $h_p$ | [m] | Pyrometer position |
| $h_0$ | [m] | Initial height of the specimen |
| $h_1$ | [m] | Height of the specimen after forming |
| $\Delta h$ | [m] | Difference in height |
| $k$ | [W/m°C] | Thermal conductivity |
| $k_c$ | [W/m$^2$°C] | Contact conductance |
| $k_f$ | [W/m°C] | Thermal conductivity of the fluid |
| $l_0$ | [m] | Initial length |
| $l_1$ | [m] | Length after forming |
| $m$ | [-] | Johnson-Cook parameter: Thermal softening coefficient |
| $m_f$ | [-] | Friction factor |

| | | |
|---|---|---|
| $n$ | [-] | Johnson-Cook parameter: Strain hardening exponent |
| $p$ | [N/m$^2$] | Pressure |
| $q$ | [W] | Heat rate |
| $q''$ | [W/m$^2$] | Heat flux |
| $q''_{rad}$ | [W/m$^2$] | Radiation heat rate |
| $s_r$ | [-] | Upset ratio |
| $s_{end}$ | [m] | Die position at the end |
| $s_{start}$ | [m] | Die position at the beginning |
| $t$ | [s] | Time |
| $t_{end}$ | [s] | Time at the end |
| $t_h$ | [s] | Heating time |
| $t_r$ | [s] | Rest time |
| $t_{start}$ | [s] | Time at the beginning |
| $t_t$ | [s] | Transport time |
| $t_1$ | [s] | Time point 1 |
| $t_2$ | [s] | Time point 2 |
| $t_3$ | [s] | Time point 3 |
| $\Delta t$ | [s] | Time difference |
| $v$ | [m/s] | Velocity |
| $v_\infty$ | [m/s] | Velocity of the fluid |

## Greek Symbols

| **Symbol** | **Unit** | **Description** |
|---|---|---|
| $\varepsilon$ | [-] | Strain |
| $\varepsilon_h$ | [-] | Strain in direction of height of the specimen |
| $\varepsilon_i$ | [-] | Strain for the i-th forming operation |
| $\varepsilon_{f\,pl}$ | [-] | Plastic strain at failure |
| $\varepsilon_{pl}$ | [-] | Plastic strain |
| $\varepsilon_s$ | [-] | Emissivity of a real surface |
| $\varepsilon_{total}^{pl}$ | [-] | Total plastic strain |
| $\dot{\varepsilon}_0$ | [s$^{-1}$] | Reference strain rate |
| $\dot{\varepsilon}_p$ | [s$^{-1}$] | Strain rate |
| $\Delta\varepsilon_{pl}$ | [s$^{-1}$] | Increment of the plastic strain |

| | | |
|---|---|---|
| $\mu$ | [-] | Friction coefficient |
| $\nu$ | [-] | Poisson ratio |
| $\rho$ | [kg/m$^3$] | Density |
| $\sigma$ | [N/m$^2$] | Stress |
| $\sigma_f$ | [N/m$^2$] | Flow stress |
| $\sigma_k$ | [W/m$^2$°C$^4$] | Stefan Boltzmann constant |
| $\sigma_N$ | [N/m$^2$] | Normal stress |
| $\sigma_I$ | [N/m$^2$] | Principal stress, direction 1 |
| $\sigma_{II}$ | [N/m$^2$] | Principal stress, direction 2 |
| $\sigma_{III}$ | [N/m$^2$] | Principal stress, direction 3 |
| $\tau_f$ | [N/m$^2$] | Shear flow stress |
| $\tau_R$ | [N/m$^2$] | Frictional shear force |
| $\varphi$ | [-] | True strain |
| $\varphi_h$ | [-] | True strain in direction of height of the specimen |
| $\varphi_i$ | [-] | True strain for the i-th forming operation |
| $\varphi_{total}$ | [-] | Total true strain |
| $\varphi_1$ | [-] | True strain in main direction (1) |
| $\varphi_2$ | [-] | True strain in main direction (2) |
| $\varphi_3$ | [-] | True strain in main direction (3) |
| $\dot{\varphi}$ | [s$^{-1}$] | Strain rate or deformation rate |
| $\dot{\varphi}_1$ | [s$^{-1}$] | Strain rate or deformation rate (index 1) |
| $\dot{\varphi}_2$ | [s$^{-1}$] | Strain rate or deformation rate (index 2) |
| $\omega$ | [-] | Damage parameter |

# List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| BBM | Black Box Modelling |
| BDA | Big Data and Analytics |
| CPS | Cyber Physical System |
| CPPS | Cyber Physical Production System |
| DAQ | Data Acquisition |
| DM | Digital Model |
| DS | Digital Shadow |
| DT | Digital Twin |
| FE | Finite Element |
| FEA | Finite Element Analysis |
| FEM | Finite Element Method |
| GBM | Grey Box Modelling |
| GUI | Graphical User Interface |
| HMI | Human Machine Interface |
| IoT | Internet of Things |
| IIoT | Industrial Internet of Things |
| LDVT | Linear Variable Differential Transformer |
| I4.0 | Industry 4.0 |
| MF | Chair of Metal Forming |
| ML | Machine Learning |
| ODB | Output Database |
| PM | Predictive Maintenance |
| RL | Reinforced Learning |
| SL | Supervised Learning |
| UL | Unsupervised Learning |
| WBM | White Box Modelling |

# 1. Introduction

As today`s globalized economy is characterized by the need for high-quality products, product customization, increasing process efficiency, process automation and a faster time-time-to market, this causes new business challenges to arise. Consequently, this leads companies to adapt to upcoming demands [1, 2]. Due to the fourth industrial revolution, also known as Industry 4.0 (I4.0), the industrial environment has undergone a significant change in recent years. The target of I4.0 is to combine the latest communication and information technology with traditional production processes leading to an increase in efficiency regarding energy and resources as well as competitiveness. Technologies like Artificial Intelligence (AI), Cyber Physical Systems (CPS), Internet of Things (IoT), Simulation and Modelling, and Big Data and Analytics (BDA) can be named as enabling technologies [1, 3, 4].

Likewise, major progress happened in simulation methods. Increasing computational capacity within the last decades enabled the use of more complex numerical methods for solving practical engineering problems [5]. Simulations are not exclusively used in an academic field, but rather became a standard tool applied in the industry with a variety of application purposes. For instance, simulations support decision making or are used to validate and test systems along the entire life cycle [3, 6]. Moreover, simulations play a significant role in realization of Industry 4.0. According to [4], simulations are a key technology of I4.0, contributing to the development and deployment of other enablers as well. Furthermore, simulations are used for process design and optimization. Additionally, in the logistics sector material flow simulations can be adapted to support decision making. Learning factories or training centers use simulations to educate people as increasing their knowledge leads to a better understanding of systems or processes and therefore reduces human errors [4].

The goal of this thesis is to implement FEA into two CPPSs, which are represented by an industrial furnace and a hydraulic press, located at the Chair of Metal Forming. Within this work, simulations to represent the whole process of upsetting of a preheated cylindrical specimen are developed. By using Python scripts, the simulation models are automatically generated and submitted to the solver. Furthermore, simulation results are evaluated, and relevant information is saved in a separate file. For the validation of the FEA, experiments were conducted and compared to the sensor data of the process. Further, a concept, to process the sensor data of the CPPSs and compare the sensor data to the extracted simulation result, is introduced and implemented.

In chapter two, a summary on state-of-the-art research is given. In the third chapter, the fundamentals are illustrated to provide basic knowledge. Subsequently, in chapter four the experimental setup is outlined, and the sensor data, provided by the CPPSs, is analyzed. Chapter five deals with the development of the simulations and provides an overview on the material properties and other parameters that are used. Further, chapter six introduces the scripting of the simulations. In Chapter seven the concept and the implementation are discussed. Finally, the results and evaluations are presented in chapter eight. A summary of the work is provided in chapter nine, also an outlook is given.

# 2. State of the art

Simulation and modelling can be named as key technology in I4.0. It describes the use of models, to improve the knowledge of the model or to make predictions of the model behavior. Thereby, a model can be either a real or imaginary system or process [6].

There are many literature sources, for example [7–9], dealing with the upsetting of a cylindrical specimen, because of the simple geometry of the model, that is easy to set up. In [10], upsetting simulation models, using different material constitutive equations and different thermal effects, are compared with each other and validated by experiments. The upsetting of a steel billet is simulated in [11] by using the Software Deform. Thereby, plastic deformation dependent on the temperature in the billet is analyzed.  Instead of a constant temperature at the beginning of the upsetting simulation, the work includes the temperature distribution after previous process steps. It accounts the heat loss during transport from the furnace to the press and also the heat loss during the contact time to the bottom die prior to the forging process [11]. In [12] a fully coupled thermomechanical analysis is conducted to simulate the forging of a spur gear in three process steps by using the Abaqus explicit solver. Heat transfer due to conduction, radiation and conductance to the tools are included. Velocity boundary conditions, accounting for the real crank movement of the press, are applied. A time- and temperature-dependent constitutive material law in combination with ductile failure criterion are accounted in the simulation to describe material behavior [12]. Predicting the flow stress of a material depending on temperature, strain, and strain rate, is crucial for the simulation of hot deformation processes. In [13] a Finite Element (FE) model coupled with a neural network is developed to model nonlinear material behavior of metals subjected to large plastic deformation at elevated temperature. Therefore, flow stress during forging operation is predicted by the neural network [13]. Further literature analyses the impacts of temperature and strain rate on the microstructure evolution during upsetting by using the thermo-mechanical coupled Finite Element Method (FEM) [14].

Simulations started as a technology, limited to very few application purposes, and developed to a standard tool, used in engineering. Establishing simulations, that include the whole life cycle of a product, is the next step in the simulation and modelling approach, which refers to the concept of a Digital Twin [15].

# 3. Fundamentals

In this chapter the fundamentals, which are necessary for this thesis, are evaluated. First, the forming technology is described, as it is crucial to understand the mechanisms taking place during forming processes. A further section focuses on the specimen material Aluminum EN AW-6082, which is used in this work. Furthermore, the phenomena of heat transfer are described. Subsequently, the focus is set on the change of industrial environment due to the fourth industrial revolution. Finally, characteristics of the numerical process simulation with the Finite Element Method (FEM) are outlined, whereas a focus is on simulations in metal forming and applied material models.

## 3.1. Forming technology

There are six main groups of manufacturing processes named forming, shaping, joining, coating, shearing, and modifying material properties, as shown in Figure 1. Further classifications consider the stress state and divide the forming process into tensile/ compressive forming, forming by pressure, forming by shearing, forming by bending and forming by tensile forces. Regarding the shape of the part to be transformed, the forming process can be divided into bulk forming and sheet forming. During sheet forming processes the part is subjected to tensile stresses and there is no significant change of the thickness of the sheet while in bulk forming processes the part is commonly subjected to compressive stresses and is three-dimensionally formed [16–18].

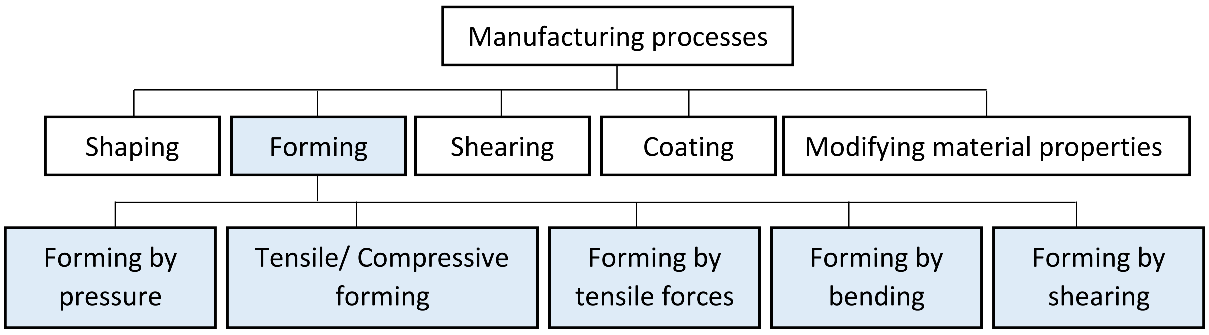


**Figure 1:** Manufacturing processes [18]

Metal forming in general is characterized as plastic deformation of a solid body under conservation of mass and material cohesion to create a product. Plastically deformed parts are shaped permanently while elastic deformations disappear when the applied force is removed. In the following characteristic parameters are evaluated [18].

By applying force to a solid body, deformations occur. Thereby, one can distinguished between strain $\varepsilon$ and true strain $\varphi$. The strain describes the change of dimension related to the initial dimension of a part. Assuming the height of a rectangular solid is reduced, the strain $\varepsilon_h$ is calculated by the height difference $\Delta h$ divided by the initial height $h_0$ of the solid body, whereby $\Delta h$ is the difference between the initial height and the height after forming $h_1$. [18]

$$\varepsilon_h = \frac{h_0 - h_1}{h_0} = \frac{\Delta h}{h_0} \tag{3.1}$$

The true strain $\varphi_h$ is defined as the natural logarithm of the height of the rectangular solid after the forming process $h_1$ divided by the initial height $h_0$ [18].

$$\varphi_h = ln \frac{h_1}{h_0} \tag{3.2}$$

For both, the elongation and the true strain, a positive value indicates an increase in dimension whereas a negative value indicates a decrease in dimension. The total true strain $\varphi_{total}$ does not depend on the sequence of forming operations unlike the total plastic strain $\varepsilon_{total}^{pl}$ [18].

$$\varphi_{total} = \sum_{i=0}^{n} \varphi_i \tag{3.3}$$

$$\varepsilon_{total}^{pl} \neq \sum_{i=0}^{n} \varepsilon_i \tag{3.4}$$

During the forming process, the volume $V$ stays constant. For a rectangular solid with an initial volume $V_0$, defined by the initial length $l_0$, height $h_0$, and width $b_0$, volume constancy is defined as

$$V_0 = V_1 = \ l_0 \cdot h_0 \cdot b_0 = l_1 \cdot h_1 \cdot b_1 = const. \tag{3.5}$$

whereas $V_1$, $l_1$, $h_1$ and $b_1$ are the volume, length, height, and width after the forming process. As a result of the volume consistency, the three values for the deformation in the main directions $\varphi_1, \varphi_2, \varphi_3$ to sum up to zero [18].

$$\frac{l_1 \cdot h_1 \cdot b_1}{l_0 \cdot h_0 \cdot b_0} = 1 \tag{3.6}$$

$$ln\left(\frac{l_1 \cdot h_1 \cdot b_1}{l_0 \cdot h_0 \cdot b_0}\right) = ln\left(\frac{l_1}{l_0}\right) + ln\left(\frac{h_1}{h_0}\right) + ln\left(\frac{b_1}{b_0}\right) = \ln(1) = 0 \tag{3.7}$$

$$\varphi_1 + \varphi_2 + \varphi_3 = 0 \tag{3.8}$$

The strain rate or deformation rate $\dot{\varphi}$ is defined as the time derivative of the true strain [18]:

$$\dot{\varphi} = \frac{d\varphi}{dt} \tag{3.9}$$

### 3.1.1. Flow stress

The flow stress curve depicts the relationship between the flow stress $\sigma_f$, also called true stress, and true strain $\varphi$. The flow stress characterizes the material behavior during plastic deformation and depends on the forming temperature, strain, strain rate, and material. As the temperature increases, the flow stress of the material decreases, which can be seen in Figure 2. Consequently, the flow stress in hot forming operations is lower than in cold forming. Furthermore, this leads to lower forming loads and higher formability, referring to the plastic deformation a material can withstand without fracture. The strain rate shows minimal effect on the flow stress in cold forming. In contrast, in hot forming the flow stress increases if the recrystallization rate increases [18, 19].

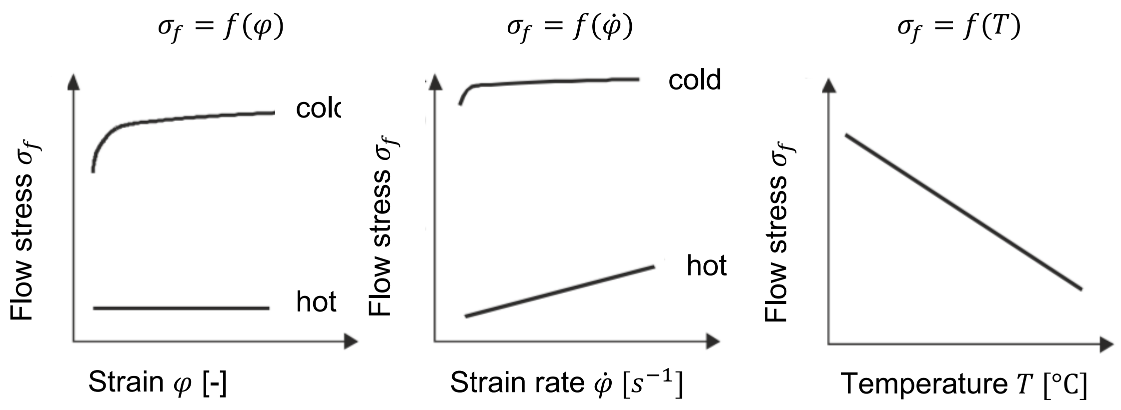


**Figure 2:** Dependence of the flow stress on the temperature [18]

There are various methods to record flow stress curves. The upsetting of a cylindrical specimen between two flat dies is a commonly used method to obtain the data for bulk forming processes. The specimen needs to keep the cylindrical form during the whole forming step, to exactly measure the true strain [18].

### 3.1.2. Cold forming and hot forming

In cold forming a specimen is formed at a forming temperature below the recrystallization temperature $T_R$, whereas in hot forming the part is preheated to temperatures above the recrystallization temperature of the material. Considering both processes, cold forming has the following advantages compared to hot forming: Manufacturing of the dies is more cost-efficient, additionally there are no costs for heating the specimen. The strength of the specimen is increased due to work hardening, additionally, there is a good surface finish and no shrinkage. The deformation rate has less impact on the flow stress. However, the cold forming process requires higher forces and has limited formability, which refers to the amount of plastic deformation, that a material can withstand without occurring fracture. The formability depends on material, forming temperature, deformation rate, and stress state. Semi hot forming is conducted at higher temperatures than cold forming, thus at lower temperatures than hot forming and therefore combines the advantages of cold forming, like work hardening, good surface finish and low tolerance range, with the high formability of a hot forming process [20].

### 3.1.3. Upsetting

Upsetting, which is a very important bulk forming process, can be classified as forming by pressure. The specimen is formed by compression in axial direction between flat dies. As the height of the part is reduced, consequently the dimensions perpendicular to the acting force increase, like demonstrated for a cylindrical specimen in Figure 3 [18].

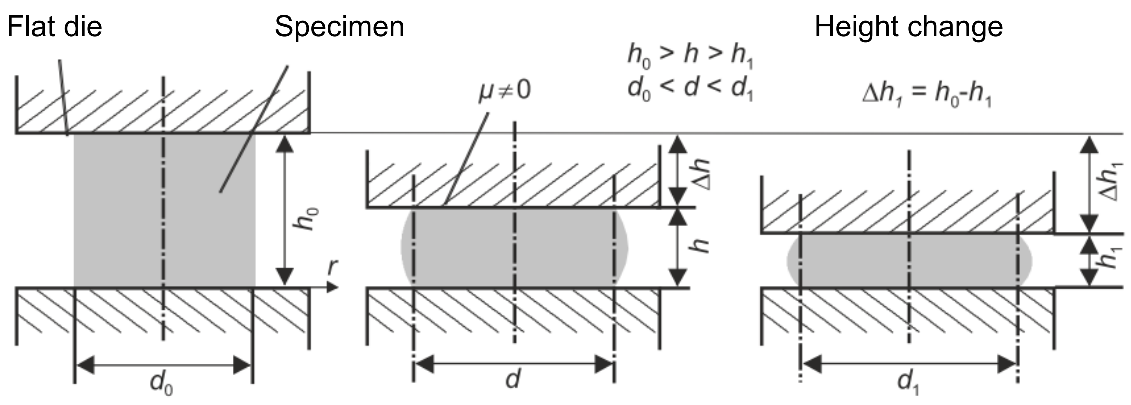


**Figure 3:** Upsetting of cylindrical part [18]

A high upset ratio $s_r$, defined as the initial height $h_0$ divided by the initial diameter $d_0$ of a cylindrical specimen, leads to buckling of the material. Therefore, the upset ratio should not exceed a certain limit of $s_r = 1{,}8 - 2{,}0$ for upsetting between flat dies [18].

$$s_r = \frac{h_0}{d_0} \tag{3.10}$$

When exceeding the material formability during forming, cracks occur. This can be avoided by either performing the upsetting process in several steps including intermediate annealing or forming at higher temperatures which leads to lower flow stresses. Furthermore, forming under hydrostatic pressure increases the formability. Due to the friction between die and specimen the actual specimen shape deviates from the ideal cylindrical shape. Radial deformation of the contact face between die and specimen is restricted by friction leading to a convex shape of the part after the forming process. To keep the cylindrical form of the specimen, lubricants or upsetting specimen with lubrication pockets in the contact surface, so called Rastagaev specimen, are used. Thereby, dimensions of the lubricant pockets need to be specified in a way that the radial force acting on the reduced contact area is at equilibrium with the frictional force [18].

### 3.1.4. Friction

Friction has great impact on metal forming processes. Forming loads and stresses in the dies increase with higher friction. Additionally, friction has an influence on the specimen surface quality. Lubricant films reduce wear of the dies as friction is reduced or specimen and die are fully or partially separated [17]. For analytical or numerical calculations of stresses, strains and forces, a mathematical formulation of the contact between specimen and die is necessary. Friction forces depend on material properties, temperature, lubrication, relative velocity between the friction interfaces, surface modification and loads, which should be considered by friction laws [18].

Friction laws commonly applied in metal forming are the Coulomb's Friction Model and the Tresca Friction Model. Whereas the Coulomb Friction Model appropriately describes the friction in case of low contact pressure, the Tresca Friction Model is suitable in case of high contact pressure like in closed-die forging or extrusion. A third friction model, a combination of both, is not further discussed

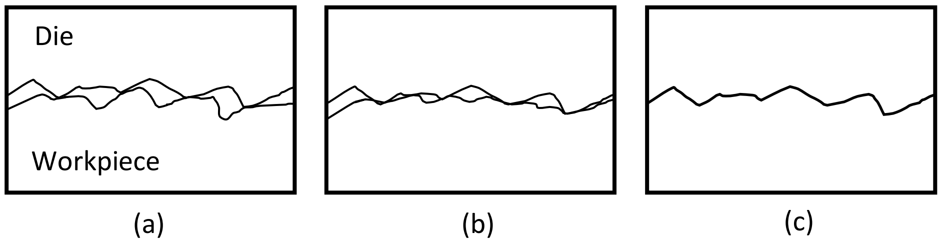


**Figure 4:** Contact interaction (a) low pressure - contact through asperity peaks (b) moderate pressure - partial conformity (c) high pressure - full conformity [17]

[17]. Depending on the level of contact stress, interactions between specimen and die vary, as shown in Figure 4. At low contact pressure, specimen and die contact each other only through highest asperity tips, which is why the real contact area is rather small. However, local plastification of asperity peaks might occur. To appropriately describe the friction in this case, the Coulomb's Friction Law is used. According to Coulomb's Friction Law, the frictional force $F_R$ is proportional to the normal acting force $F_N$, respectively the frictional shear force $\tau_R$ is proportional to the normal stress $\sigma_N$. Thereby, the friction coefficient $\mu$ is the proportional factor [17].

$$F_R = \mu \cdot F_N \tag{3.11}$$

$$\tau_R = \mu \cdot \sigma_N \tag{3.12}$$

At high contact pressure large plastic deformations of the softer contact body occur, which squeezes the softer material into the roughness valleys of the die. Specimen and die contact each other over the whole area. Friction stress cannot exceed the shear flow stress $\tau_f$. If the shear flow stress is reached, no sliding in the interface between specimen and die occurs. At high contact pressure the Tresca Friction Model is used, which is defined as [17]

$$\tau_R = m_f \cdot \tau_f \tag{3.13}$$

including the friction factor $m_f$, which varies in the range $0 < m_f < 1$ and the shear flow stress $\tau_f$. The friction factor is equal to 1 if specimen and die stick together, the factor is equal to 0 for the frictionless case. Coloumb and Tresca friction models are depicted in Figure 5 [17].

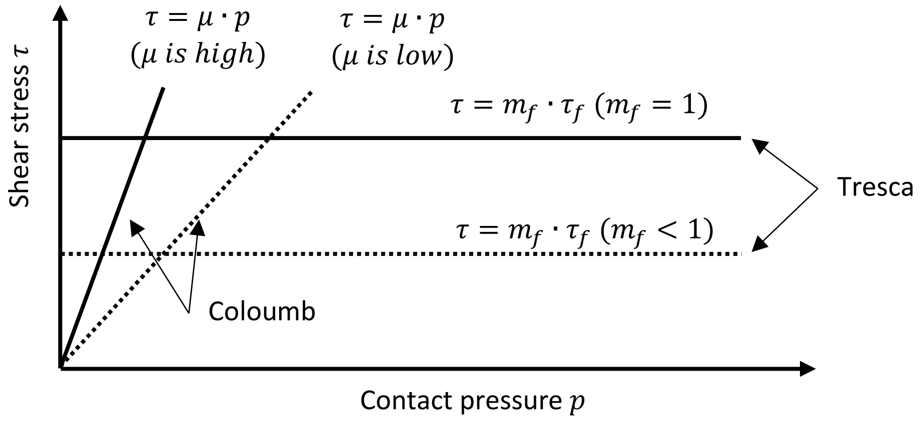

$\tau = \mu \cdot p$ ($\mu$ is high)
$\tau = \mu \cdot p$ ($\mu$ is low)
$\tau = m_f \cdot \tau_f$ ($m_f = 1$)
Tresca
$\tau = m_f \cdot \tau_f$ ($m_f < 1$)
Coloumb
Shear stress $\tau$
Contact pressure $p$

**Figure 5:** Friction models [17]

### 3.1.5. Thermal effects during forming

Heat radiation causes a significant loss of heat of the specimen at temperatures above 1000-1200°C. At low temperatures the influences due to heat radiation are negligible, for instance for forming temperatures for aluminum alloys which are below 550°C. Without consideration of heat radiation and convection to the environment, the temperature of a specimen $T$ is described as follows, whereas $T_0$ is the initial temperature of the specimen [17]:

$$T = T_0 + \Delta T_D + \Delta T_F - \Delta T_T \tag{3.14}$$

In this equation $\Delta T_D$ expresses the temperature increase of the specimen due to the dissipated deformation energy. As sliding occurs in the interface between specimen and die, energy dissipates causing temperature to increase which is described by the term $\Delta T_F$. Considering hot forming the initial die temperature is much lower than the temperature of the specimen, causing heat to transfer from the specimen to the die. At the initial state of cold forming applications, die and specimen, are at room temperature. As the specimen is heated while forming, heat transfers to the die. The decrease of temperature due to heat conduction to cooler dies is considered as $\Delta T_T$ [17].

### 3.1.6. Process parameters

As the mechanical properties of the product after the forming process depend on the conditions during the forming process, it is crucial to measure and control the entire process. Important quantities, for instance shown in Figure 6, are the flow stress $\sigma_f$, the strain rate $\dot{\varepsilon}$, the strain $\varepsilon$, the temperature $T$, the shear stress $\tau$ and the contact pressure $p$. To undergo the intended plastic deformation without fraction, the formability of the material is important. Additionally, the lubrication has an impact on the process parameters [17].

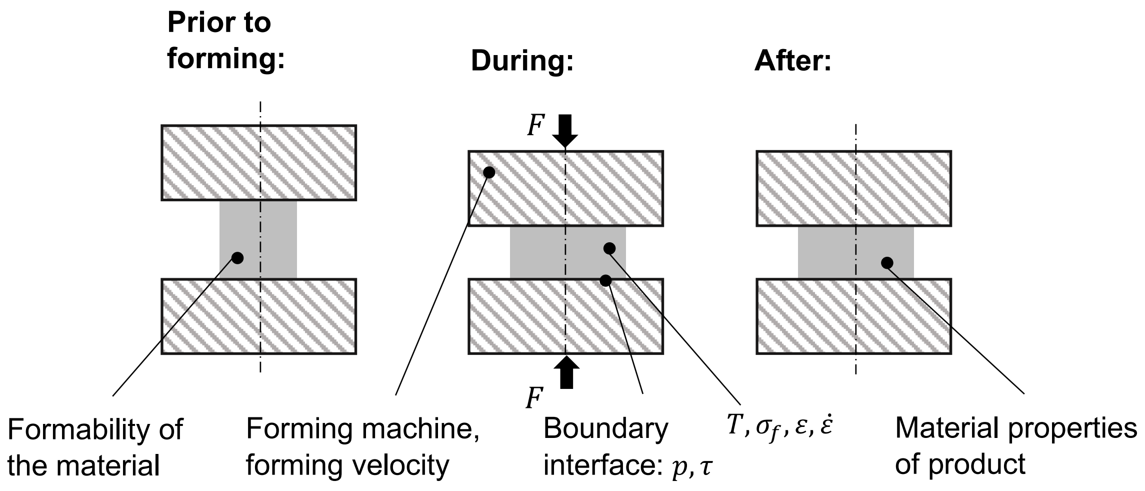

Prior to forming:
During:
After:
$F$
$F$
Formability of the material
Forming machine, forming velocity
Boundary interface: $p, \tau$
$T, \sigma_f, \varepsilon, \dot{\varepsilon}$
Material properties of product

**Figure 6:** Relevant process parameters [17]

## 3.2. Aluminum alloys

DIN EN 573 and DIN EN 1780 divide aluminum alloys into two main groups [21]: wrought alloys and casting alloys. Wrought alloys are preformed to bars or tapes in continuous casting processes and previously manufactured to rolled, pressed, and drawn products. Casting alloys are characterized by good mold filling properties and insensitivity to hot cracking and therefore are used in casting processes. Standardized designation of aluminum alloys includes the prefix EN followed by the letter A - for aluminum. The next letter denotes the manufacturing either as W for wrought alloy or C for casting alloy. The alloy composition is defined by the following four numbers for wrought alloys or five letters for casting alloys. For wrought alloys the first number defines the alloy group characterized by one or more main alloy elements. The last two numbers are characteristic for the specific alloy or define the degree of purity (e.g., Al99.5 = 1050, Al99.7 = 1070) for group 1XXX (pure aluminum), which contains a mass percentage of 99.0 % to 99.9 % of aluminum. Examples for the standardized designation of aluminum wrought alloys are given in Table 1 [21].

**Table 1:** Standardized designation of aluminum wrought alloys [21]

| Group | Alloy type | Example |
|---|---|---|
| 1XXX | Pure aluminum | EN AW-1050A |
| 2XXX | AlCu | EN AW-2024 |
| 3XXX | AlMn | EN AW-3003 |
| 4XXX | AlSi | EN AW-4046 |
| 5XXX | AlMg | EN AW-5182 |
| 6XXX | AlMgSi | EN AW-6082 |
| 7XXX | AlZnMg | EN AW-7020 |
| 8XXX | Other | EN AW-8011A |

The material used for the cylindrical specimen in the practical experiments is the aluminum alloy EN-AW-6082 with the chemical composition specified in Table 2.

**Table 2:** Chemical composition EN-AW-6082 [21]

| Alloy elements % per weight | | | | | | |
|---|---|---|---|---|---|---|
| **Si** | **Fe** | **Cu** | **Mn** | **Mg** | **Cr** | **Ti** |
| 0.7- 1.3 | 0.5 | 0.1 | 0.4-1 | 0.6-1.2 | 0.25 | 0.1 |

## 3.3. Heat transfer

In the following, the mechanisms of heat transfer - conduction, convection, and radiation - are defined. Furthermore, a 0-dimensional transient heat conduction problem is outlined.

### 3.3.1. Conduction

A temperature gradient through a solid material causes heat to conduct from the high-temperature site to the lower temperature site. Fourier's Conduction Law for a one-dimensional conduction problem, like in Figure 7, is defined as [22]

$$q'' = -k\frac{dT}{dx} = k\frac{T_1 - T_2}{L} \tag{3.15}$$

$$q'' = \frac{q}{A_c} \tag{3.16}$$

including the heat flux $q''$, the heat rate $q$, the thermal conductivity $k$ of the solid material, the cross-sectional area $A_c$, the temperatures $T_1$, $T_2$ and the conduction length $L$ . To determine the temperature profile in case of heat conduction the thermal conductivity, density and specific heat of a material should be given [22].

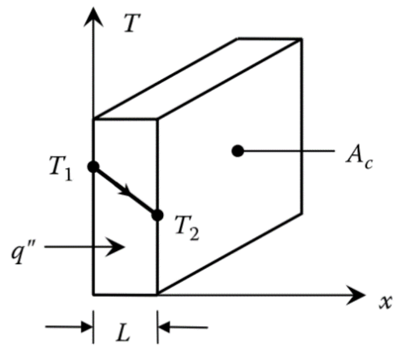


**Figure 7:** 1-D conduction through a wall [22]

### 3.3.2. Convection

Fluid or gas flow over a solid surface causes convection, for instance, Figure 8 demonstrates the cooling of a heated surface due to air flow. The heat removal rate from the heated surface is proportional to the difference between the temperature of the fluid $T_\infty$ and the surface temperature at the wall $T_S$. Thereby the proportional constant $h_c$ is the heat transfer coefficient. Applying Fourier Conduction Law to the cooling fluid, the same heat rate can be determined. In the following equations, $k_f$ is the thermal conductivity of the fluid, $A_S$ the surface area for convection. The heat transfer coefficient is influenced by fluid properties, flow conditions, surface configurations and others. It can be differentiated between natural convection and forced convection [22].

$$q'' = h_c(T_S - T_\infty) = -k_f \frac{dT}{dy} \tag{3.17}$$

$$q'' = \frac{q}{A_S} \tag{3.18}$$

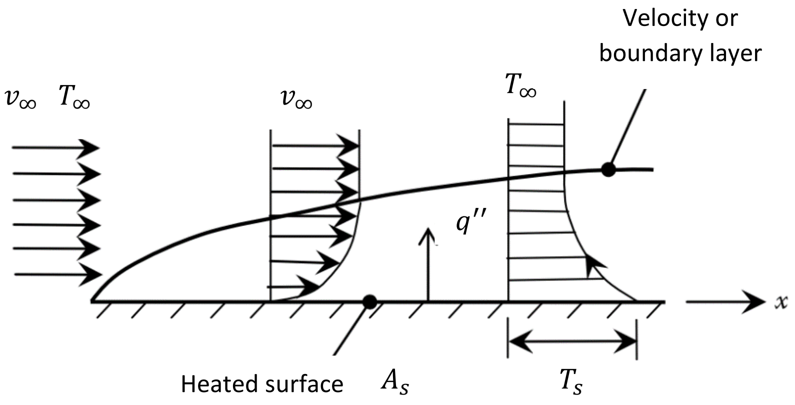


**Figure 8:** Convection [22]

### 3.3.3. Radiation

Solids, liquid surfaces, or gases at temperatures higher than absolute zero cause electromagnetic waves that transfer heat, as illustrated in Figure 9. Radiation heat rate $q''$ is defined by the Stefan-Boltzmann Law [22].

$$q''_{rad} = \varepsilon_S \sigma_k {T_S}^4 \tag{3.19}$$

$$q''_{rad} = \frac{q}{A_s} \tag{3.20}$$

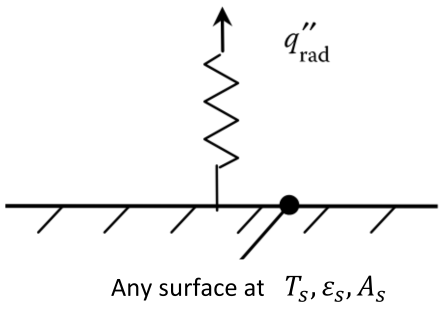


**Figure 9:** Radiation from a solid surface [22]

The parameter $\varepsilon_s$ is the emissivity of the real surface, $\sigma_k = 5.67 \cdot 10^{-8}$ the Stefan Boltzmann constant, $T_S$ the surface temperature and $A_s$ the surface area for radiation. The emissivity of a surface is between 0 and 1, whereas the emissivity for an ideal (black) surface $\varepsilon_s = 1$. In general, the emissivity depends on material, temperature, and wavelength [22].

### 3.3.4. Transient heat transfer

A transient heat transfer problem is characterized by the change in temperature of a solid material with location as well as with time. Assuming the temperature of an object changes uniformly and depends only on the time, some real time applications can be modeled as zero dimensional (0-D) problems. Applying the lumped capacitance method can solve this special case of a 0-D transient heat transfer problem. From the energy balance on a solid material with density $\rho$, volume $V$ and specific heat capacity $c$ follows [22]:

$$\rho V c \frac{dT}{dt} = q_s{}'' A_{s,h} - h_c\,(T - T_\infty) A_s - \varepsilon_S \sigma_k \left(T^4 - {T_\infty}^4\right) A_s \tag{3.21}$$

This equation considers a heat flux $q_s{}''$ applied on a portion of the surface area $A_{s,h}$, convection and radiation. To obtain an approximate solution of this first-order, nonhomogeneous, ordinary differential equation the finite-difference method can be applied. The lumped capatictance method is valid if the entire material is assumed to uniformly change with temperature. As an approximation, the Biot (Bi) number can be calculated as [22, 23]

$$Bi = \frac{h_c \cdot L_c}{k} \tag{3.22}$$

whereby $L_c$ is the characteristic length of the material, $h_c$ defines the convection heat transfer coefficient, and $k$ is defined as the thermal conductivity of the material. Values less than 0.1 indicate the validity of this method. The approximation of the lumped capacitance method is better the smaller the Biot number is, which indicates a small geometry of a material with high conductivity and low convective cooling or heating. The characteristic length is defined by the volume of a solid body diveded by the surface area. [22, 23]

$$L_c = \frac{V}{A_s} \tag{3.23}$$

## 3.4. Industry 4.0

In 2011 the term I4.0 was introduced for the first time in Germany and since then, received attention in academic and industrial field. Industry 4.0, refers to the ongoing revolution in manufacturing environment to enhance products and production processes by using automation and digitalization technologies [3, 24]. Additionally, connections between virtual and real world are established [2], enabled for instance by the Internet of Things (IoT). Each object connected via IoT technologies interacts with others, which allows interactions between machines, products, tools etc. leading to intelligent processes and an increase in efficiency [25]. Key technologies of Industry 4.0 include Cyber-Physical Systems (CPS), Internet of Things (IoT), Big Data and Analytics (BDA), Cloud computing, Artificial Intelligence (AI), Augmented Reality, Simulation and Modelling, Visualization Technology, Cybersecurity and Automation and Industrial robots. [1, 6]

Sensors and the respective data acquisition (DAQ) systems acquire data of manufacturing processes. Human-machine interfaces (HMIs), such as touch panels, keyboards, or switches, enable access to processed sensor signals and allow humans to interact and input commands [26]. CPSs are a key element in I4.0 as they enable the connection between the virtual and the physical world. The term Cyber Physical Production Systems (CPPSs) refers to CPSs applied in the production environment. CPPSs can be described as systems of systems capable of complex interactions due to the connections among autonomous and cooperative elements. CPPSs are able to adapt to varying conditions during the whole production lifecycle, improve real-time decision-making or autonomously fulfill cognitive tasks [27].

A Digital Twin (DT) refers to a virtual representation of a real physical product. Considering the exchange of data, it can be further distinguished between Digital Twin (DT), Digital Shadow (DS) and Digital Model (DM). A DT is characterized by bilateral automatic data exchange between virtual and real entity, whereas a DS enables unidirectional, and a DM has no automatic data transfer. There are two general approaches to generate data that is necessary for modelling a DT, DS or DM. While the White Box Modelling (WBM) approach uses real-physical laws, the Black Box Modelling approach uses stochastic methods based on process and sensor data. The combination of both approaches (WBM and BBM), called Grey Box Modelling (GBM), gains popularity and additionally provides great potential for future applications of DS and DT in the metal forming industry [24].

Data is essential for all I4.0 technologies. Big Data concepts use huge amount of data to raise economic value [25]. Generated data of production processes needs to fulfill three criteria: volume, variety, and velocity. Volume describes the amount of data generated in a digitalized factory. As there are different sources of the data, a huge variety of data occurs leading to complex data structures. Velocity refers to the speed the required data is provided at [24]. The three criteria can be further extended to five characteristics, so called 5Vs, including also veracity and value [28].

Another  I4.0 concept is the smart factory, also called digital or intelligent factory, that represents a future manufacturing system which is fully connected via the IIoT and thus capable of mainly operating without human force [29]. Predictive Maintenance (PM) is a key technology of a smart factory, aiming to predict machine failure based on machine data. Machine data acquired by sensors, is stored in a database to make the data available at any time. The results of the data analysis are used to plan maintenance [25].

Machine Learning (ML) is a subdomain of AI. In general, ML describes systems that are capable of cognitive abilities like humans. More specifically, ML is based on complex algorithms and uses data for training a model, which can further predict results. Data for ML is divided into data for training and testing. Whereas the trainings data is necessary to develop a model and the test data verifies the desired output. There are three different methods to train the model: supervised learning (SL), unsupervised learning (UL), and reinforced learning (RL). Provided input and output data for training purposes is necessary for the model to understand correlations used in further predictions, which is called SL. Input data is provided to a model which autonomously finds hidden patterns and adapt the algorithm through UL. RL uses the feedback of previous actions to improve the model. ML provides benefits for complex analysis, e.g., to control machines to enhance efficiency [25].

## 3.5. Finite Element Method (FEM)

The FEM is a numerical technique, that divides a model into a finite number of elements, to find approximate solutions of differential equations in engineering problems and physics. Shape functions are used to approximate state variables within an element. Equations of each finite element are assembled and consequently solved [30, 20].

A Finite Element Analysis (FEA) includes three steps. The first step is the preprocessing including geometry definition, meshing, definition of material properties and boundary conditions. The next step is the calculation with the solver, subsequently followed by the postprocessing which refers to the visualization and evaluation of the results [20].

There are implicit and explicit procedures. In general, the whole process is divided into time increments $\Delta t$. Implicit solvers calculate unknown variables for each time step under consideration of the values at the time $t$ as well as at the time $t + \Delta t$. At the end of each time increment the system is in equilibrium state, therefore many iterations are necessary. Explicit procedures calculate unknown parameters at the time $t + \Delta t$ using only values available at the time $t$. No equilibrium state is calculated and therefore no iterations are necessary leading to a reduction in computational effort. However, the time increment needs to be very small to minimize inaccuracies in calculation. The time increment $\Delta t$ needs to be smaller than the time an elastic wave needs to pass the distance equal to the shortest element length and therefore depends on the speed of sound. For solid bodies the speed of sound $c_0$ depends on the Young`s modulus $E$ and the density $\rho$ [20].

$$c_0 = \sqrt{\frac{E}{\rho}} \tag{3.24}$$

A linear FEA is characterized by a linear relation between applied loads and the response of the system, which is valid if the nonlinear behavior of a real physical systems is negligible. In general, the sources of nonlinearities are classified as material, geometry, initial or boundary conditions. Whether a linear or nonlinear analysis is carried out, depends on the desired outcome of the simulation and the tolerated errors. For instance, a nonlinear analysis is essential to represent the real material behavior, improve knowledge of specific phenomena, evaluate reasons of system failure, design high-performance parts (e.g., in aerospace industry) or determine functionality under damage and failure exhibition [5]. Nonlinear problems are solved iteratively, e.g., by using the Newton-Raphson-Method [18].

Heat transfer analysis procedures can be classified in uncoupled heat transfer, sequentially coupled or fully coupled thermal-stress analysis. Uncoupled heat transfer analysis is used, if the temperature field does not depend on the stress and deformation state. To conduct a sequentially coupled thermal-stress analysis, first the temperature field is calculated as a pure heat transfer problem and afterwards the temperature distribution is used as initial state in the stress analysis. Fully coupled thermal-stress analysis is required, if thermal and mechanical solution strongly influence each other, thereby stress/displacement and temperature fields are solved simultaneously [31].

Computation time is an important aspect, which depends on whether an implicit or explicit method is used. Element type and order of the shape function have an influence as well. The finer a mesh is, the higher is the computation time. However, computation time can be reduced through symmetry boundaries, or if rigid elements are used to model forming tools [32].

## 3.6. Metal forming simulations

Due to cheaper and more efficient computers, FEA became a standard industry tool used for the simulation of metal forming processes as it provides productivity and user-friendliness. Nonlinear FEM offers great potential in process design and optimization. Additionally, expensive and time-consuming experiments can be replaced [18]. Whereas general purpose FEM codes, such as ANSYS and ABAQUS, are highly flexible and can be used for various applications, special purpose FEM codes like FORGE, Q-FORM, DEFORM, and SIMUFACT FORMING, tailored for the application in bulk-metal forming, are especially user-friendly. However, general purpose FEM code often require great knowledge and are time consuming [17, 20].
Simulations for bulk-forming processes are used to determine material flow, material hardening, microstructure, formability and the mechanical, thermal and tribological loads acting on the tools [20]. Especially in metal forming, large plastic deformations, contact between specimen and tool, temperature and incompressibility have to be considered [33]. Therefore, to simulate forming processes, nonlinearities need to be considered, which leads to complex models that require high computation time. Especially in hot bulk forming processes large distortions of the finite elements occur, leading to distorted meshes. Therefore, commercial FEM software provides remeshing, to transform the state variables from the distorted mesh to the new one, which is also called rezoning. Furthermore, friction and temperature effects, like heat transfer to the environment or dies, have high importance to describe the forming process accurately [18]. To describe the material behavior of a forming process the FEM model requires flow stress data. The flow stress depends on the temperature and on the strain rate. Either a graph, providing the stress-strain data, or a mathematical function in the form of a material model are used to implement this information to the simulation model [17].

## 3.7. Material models

The material models used in metal forming simulations can be divided into two main groups, as shown in Figure 10. On the one hand, there are material models that do not consider elastic behavior and assume the material to be rigid until plastic flow occurs. The utilization of rigid-plastic material models takes less computational time and is valid for many forming applications as plastic deformations are larger than elastic deformations. On the other hand, there are material models that consider elastic and plastic behavior. This elastic-plastic material models are especially important, for instance if springback, or residual stresses need to be evaluated. Additionally, viscous models describe rate dependent behavior, which is, for example, important in hot or semi-hot forming of steels. [20]

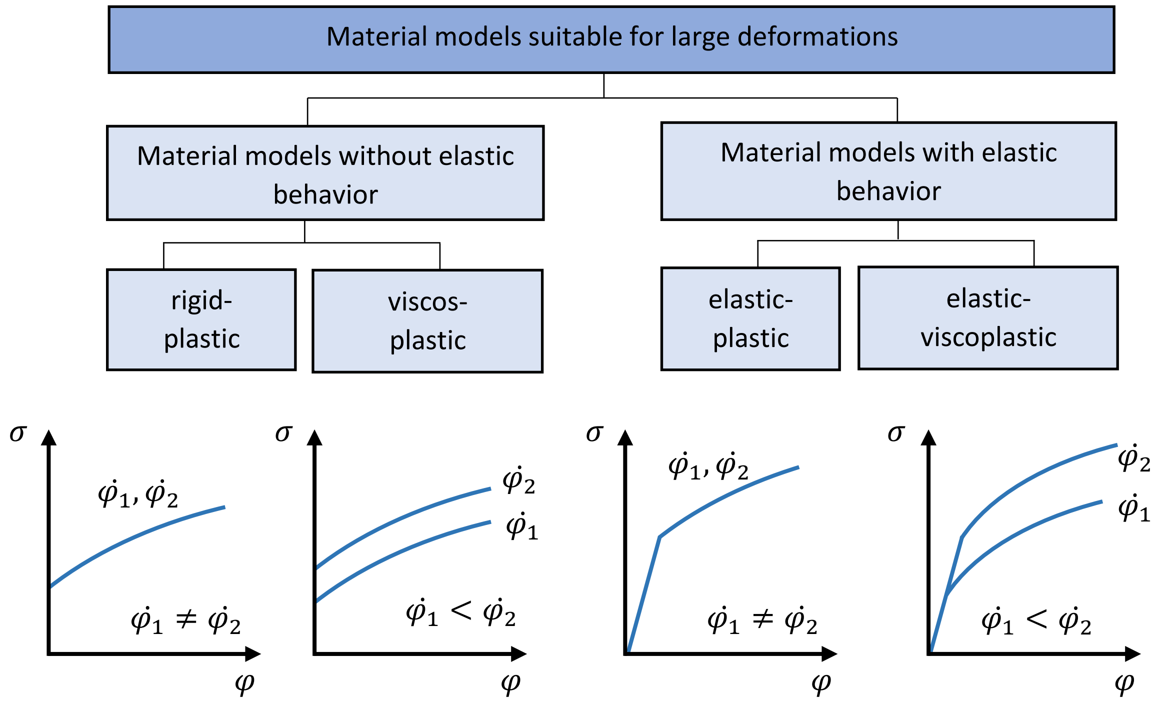

Material models suitable for large deformations
Material models without elastic behavior
Material models with elastic behavior
rigid-plastic
viscos-plastic
elastic-plastic
elastic-viscoplastic
$\sigma$
$\varphi$
$\dot{\varphi}_1, \dot{\varphi}_2$
$\dot{\varphi}_1 \neq \dot{\varphi}_2$
$\dot{\varphi}_2$
$\dot{\varphi}_1$
$\dot{\varphi}_1 < \dot{\varphi}_2$

**Figure 10:** Material models suitable for large deformations [20]

To describe plastic material behavior, the material model needs to consider yield criteria, flow rule and hardening. The yield criterion describes the onset of plastic material flow as soon as the equivalent stress reaches the flow stress. Therefore, a multiaxial stress condition is transferred to an equivalent uniaxial stress condition. The extend and direction of the plastic deformation are defined by the flow rule. The change in mechanical material properties during plastic deformation is considered by modifications in the yield criterion through hardening laws. Isotropic and kinematic hardening are depicted in Figure 11. For isotropic hardening the yield surface increases, without a change in the position. By contrast, for kinematic hardening the yield surface stays constant, whereas the position shifts in load direction. Consequently, preceding tensile loads lead to lower flow stress for compressive loads, also described as Bauschinger-Effect. kinematic and isotropic hardening represent an ideal material behavior, whereas real materials show a combination of both hardening models. Thermal softening of the material can occur due to recovery and / or recrystallization [18].

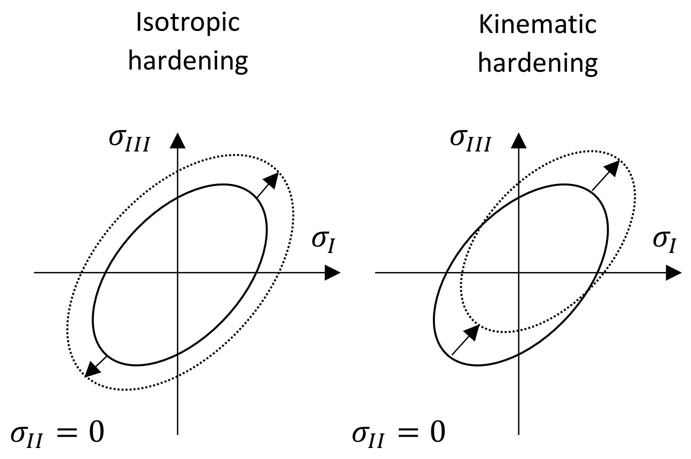


**Figure 11:** Isotropic and kinematic hardening [18]

# 4. Experimental setup

In the following, the hydraulic press and the furnace at the Chair of Metal Forming (MF) are described, as well as the sensors they are equipped with. Furthermore, an overview on the setup of the practical experiments is given, and the experimental plan is outlined. Additionally, focus is set on the automatic evaluation and visualization of the measured sensor data with Python.

## 4.1. Furnace

The furnace at the chair of metal forming, shown in Figure 12, can be heated up to maximum temperature of 1200 °C via resistance heating. The dimensions of the heating chamber of the furnace are 300 mm in width, 240 mm in height and 450 mm in depth, whereas the furnace lining is made of refractory material. The furnace is equipped with thermocouples, which measure the air temperature inside the heating chamber.

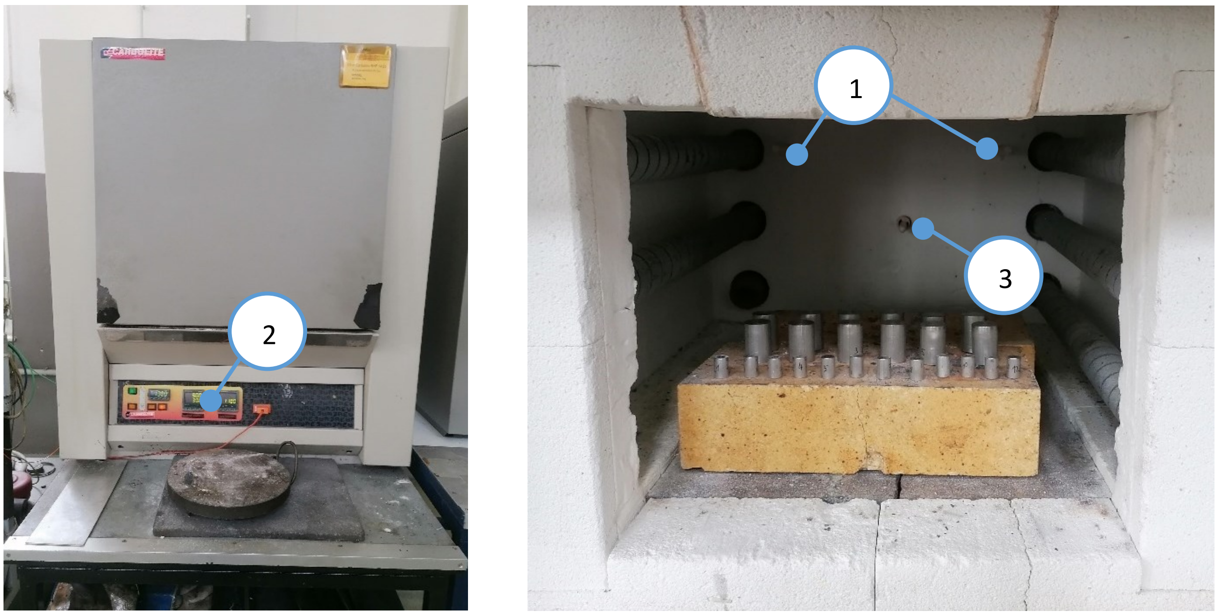


**Figure 12:** Furnace at the Chair of Metal Forming

Two thermocouples (1) are located in the upper left and the upper right corner of the back wall of the heating chamber and are used by the control system of the furnace. The measured temperature of those thermocouples is shown on the display (2) of the control system on the front side of the furnace. By applying the retrofitting method, the furnace was equipped with an additional thermocouple Type K (3), placed at the center of the back wall. The measured temperature of this thermocouple can be displayed via the implemented HMI [34].

Table 3: Sensor of the furnace [34]

| Measured quantity | Sensor | Range |
|---|---|---|
| Temperature [°C] | Thermocouple Type K | 0-1200°C |

## 4.2. Hydraulic Press

Metal forming aggregates are categorized by different working principles, related to ram displacement, applied force and provided kinetic energy [19]. Hydraulic presses are controlled by force, which can be regulated via hydraulic pressure. The nominal force of the forming unit is available during the whole stroke. Additionally, ram kinematics can be individually adjusted to the forming process. Compared to metal forming aggregates controlled by displacement, the ram velocity and therefore the ratio between production output and input is lower. [16, 18]

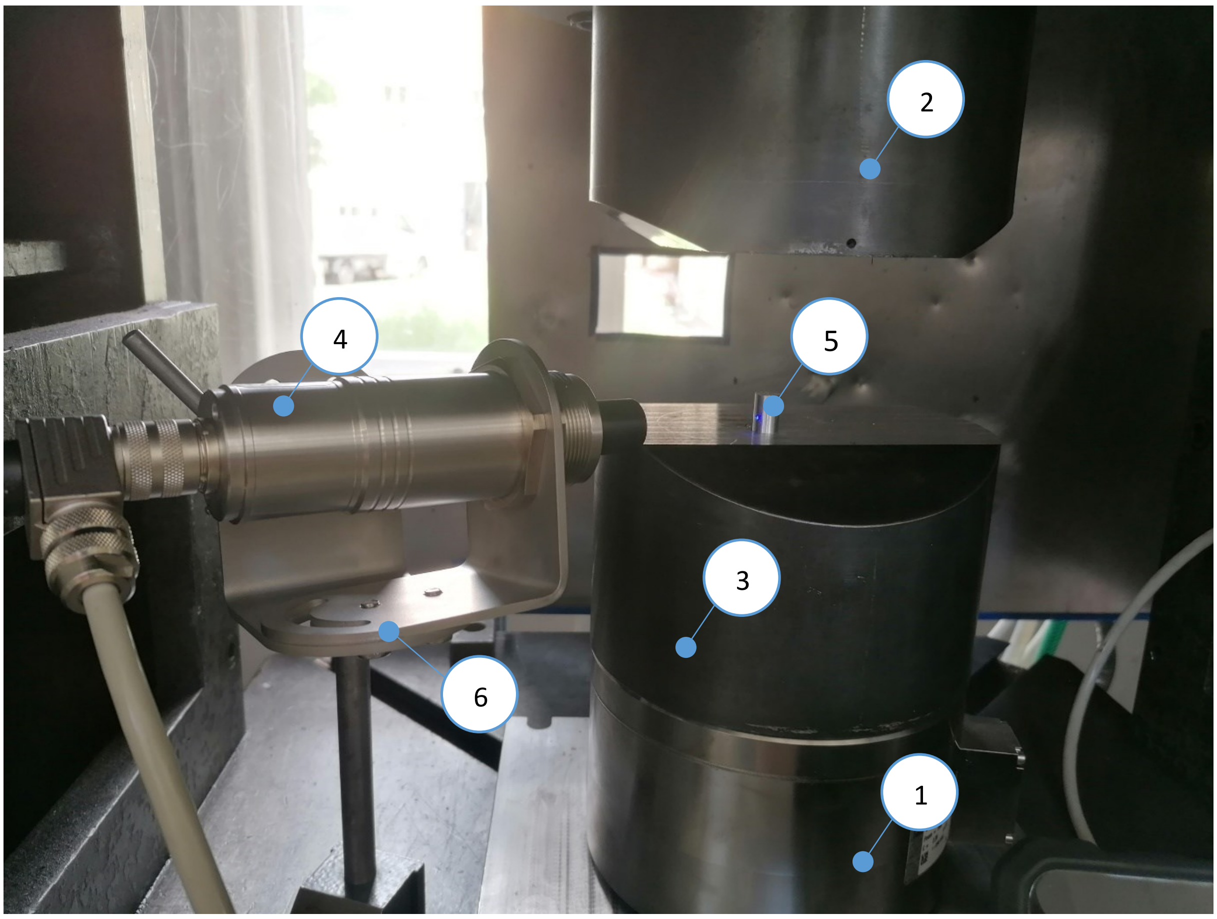


**Figure 13:** Hydraulic press at the MF

The hydraulic press at the MF, depicted in Figure 13, is located directly next to the furnace, enabling shorter transportation time, and therefore reducing temperature loss of the specimen. A load cell (1), appropriable for the maximum load of 1 MN, measures the force applied during the bulk forming process. The position of the top die (2) during the forming process is traced with a Linear Variable Differential Transformer (LVDT) with the range of 0-600 mm, whereas the bottom die (3) remains at a fixed position. Additionally, the hydraulic press was equipped with a pyrometer (4) to measure the temperature of the specimen (5) in the range of 0-1200°C. The pyrometer is attached to a mobile, height-adjustable mounting (6), enabling the modification of the pyrometer position. [35]

**Table 4:** Sensors of the hydraulic press [34]

| Measured quantity | Sensor | range |
|---|---|---|
| Die force [N] | Load cell | 0-1 MN |
| Die position [mm] | LVDT | 0-600 mm |
| Temperature of the specimen [°C] | Pyrometer | 0-1200°C |

## 4.3. Procedure

Cylindrical specimens were tested, whereby two different dimensions, listed in Table 5, were used in the experiments. For identification purpose, specimens were labelled as “A” or “B” referring to the respective specimen dimensions.

**Table 5:** Specimen dimensions

| Specimen label | Initial diameter $d_0$ [mm] | Initial height $h_0$ [mm] |
|---|---|---|
| A | 10 | 15 |
| B | 20 | 30 |

The furnace is preheated to a defined operating temperature $T_F$ before the specimen are put into the heating chamber for preheating. At the time the specimen is taken out of the furnace with a manual gripper, the measurement is started via the HMI. The data of the measured quantities, listed in Table 3 and Table 4, is automatically recorded. Additionally, the elapsed time is measured and displayed on

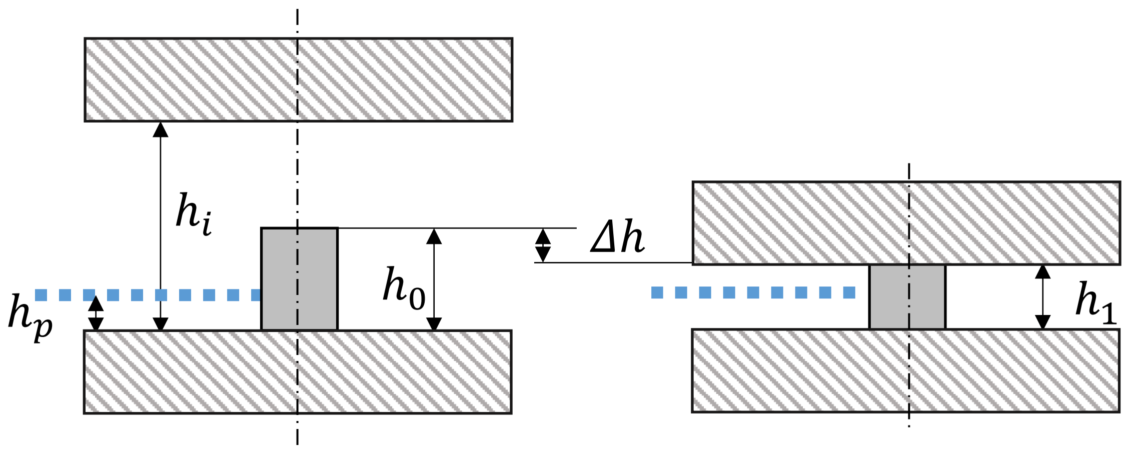


**Figure 14:** Schematic setup of the pyrometer position

the HMI. The specimen is transferred from the furnace to the hydraulic press within the specified transportation time $t_t$. As the specimen is placed on the bottom die, the pyrometer measures the temperature of the specimen at a defined position.

Thereby, the measuring position of the pyrometer $h_p$ is defined in such a way that the top die does not interfere with the measuring position during upsetting process, as shown in Figure 14. After positioning the specimen remains on the bottom die for a specified rest time $t_r$ before the hydraulic press is activated. the initial distance between top and bottom die $h_i$ of the hydraulic press is the same for each upsetting process. After the upsetting process, the specimen is removed from the hydraulic press and the measurements is manually stopped by using the HMI, as the top die reaches its original position. Measured data is stored in a csv-file. The output file contains the timestamp (formatted, e.g., T#10s300ms), the load of the hydraulic press, the absolute and relative gap between the dies, the specimen temperature measured by the pyrometer, and the temperatures measured by the thermocouples. The sequence starts again for the next specimen.

Overall, two experiments, each with different test settings, were carried out. For a specified test setting, different process parameters were defined, which are the dimension of the specimen, the furnace temperature, the transfer time and the height difference of the upsetting process. The measuring position of the pyrometer, the relative distance between top and bottom die and the rest time of the specimen on the bottom die are constant for each test setting, see Table 6.

**Table 6:** General settings for all experiments

| **Measuring position $h_p$ [mm]** | **Initial distance $h_i$ [mm]** | **Rest time $t_r$ [s]** |
|---|---|---|
| 6 | 50 | 3 |

### 4.3.1. Experiment 1

A total amount of 24 specimen per geometry were available for testing. For identification purpose, each specimen was assigned a number in combination with the label A or B specifying the dimensions as given in Table 5. Overall, eight different test settings, outlined in Table 7, were defined, whereas six specimens were tested for each setting. Furnace temperature varies on two different levels, 300°C or 500°C respectively. All specimens referring to the same operating temperature of the furnace were placed in the furnace and heated for half an hour to ensure homogeneous heating of the whole specimen. The transfer time varies between four and seven seconds.

**Table 7:** Experimental plan – experiment 1

| Setting number | Specimen number | Temperature $T_F$ [°C] | Transfer time $t_t$ [s] | Height difference $\Delta h$ [mm] |
|---|---|---|---|---|
| 1 | 1A, 2A, 3A, 4A, 5A, 6A | 300 | 4 | 5 |
| 2 | 7A, 8A, 9A, 10A, 11A, 12A | 300 | 4 | 8 |
| 3 | 1B, 2B, 3B, 4B, 5B, 6B | 300 | 4 | 15 |
| 4 | 7B, 8B, 9B, 10B, 11B, 12B | 300 | 7 | 20 |
| 5 | 13A, 14A, 15A, 16A, 17A, 18A | 500 | 7 | 5 |
| 6 | 19A, 20A, 21A, 22A, 23A, 24A | 500 | 4 | 8 |
| 7 | 13B, 14B, 15B, 16B, 17B, 18B | 500 | 7 | 15 |
| 8 | 19B, 20B, 21B, 22B, 23B, 24B | 500 | 4 | 20 |

### 4.3.2. Experiment 2

A second experiment series, similar to the first one, is carried out. Throughout the tests, special attention is paid on the measuring position of the specimen. The pyrometer is placed at the back side in the hydraulic press. The specimen is inserted into the hydraulic press, whereas it is positioned in a way that the pyrometer does not measure the temperature at the area where the gripper contacted the specimen. The experimental plan is given in Table 8. For this experiment 24 specimen of geometry A were tested in six different settings, each containing four specimens. Temperature is tested on three different levels, while the transfer time varies from four to seven seconds and the height difference is constant at five millimeters for each setting.

**Table 8:** Experimental plan – experiment 2

| Setting number | Specimen number | Temperature $T_F$ [°C] | Transfer time $t_t$ [s] | Height difference $\Delta h$ [mm] |
|---|---|---|---|---|
| 1 | 1A, 2A, 3A, 4A, | 300 | 4 | 5 |
| 2 | 5A, 6A, 7A, 8A | 300 | 7 | 5 |
| 3 | 9A, 10A, 11A, 12A | 400 | 4 | 5 |
| 4 | 13A, 14A, 15A, 16A | 400 | 7 | 5 |
| 5 | 17A, 18A, 19A, 20A | 500 | 4 | 5 |
| 6 | 21A, 22A, 23A, 24A | 500 | 7 | 5 |

## 4.4. Sensor data visualization

A Python script was used to visualize the sensor data for each measurement file saved in a specified directory. Measurements belonging to the same test setting are printed in the same diagram. A test setting is defined by the setting name, the diameter and height of the specimen, the preheating furnace temperature, the upset height, the transport time, and rest time. The setting name serves the identification purpose, all further information of the setting is displayed within the plot. To assign a measurement file to a test setting, the Python script needs information about the file names that belongs to an experiment setting. Therefore, the name of the measurement file needs to be specified manually. An example for the assignment of all measurement files to the corresponding test setting of an experiment is given in Figure 15.

```
# 1.) Setting definition:
# [setting name, diameter, height, preheating temperature, upset height, transport time, rest time]
s_1 = ['s1', 10, 15, 300, 5, 4, 3]
s_2 = ['s2', 10, 15, 300, 5, 7, 3]
s_3 = ['s3', 10, 15, 400, 5, 4, 3]
s_4 = ['s4', 10, 15, 400, 5, 7, 3]
s_5 = ['s5', 10, 15, 500, 5, 4, 3]
s_6 = ['s6', 10, 15, 500, 5, 7, 3]
settings = [s_1, s_2, s_3, s_4, s_5, s_6]  # all tested settings

# 2.) Assignment of measurement files to corresponding test setting
sp_1 = ['TestNr_12.', 'TestNr_13.', 'TestNr_14.', 'TestNr_15.']
sp_2 = ['TestNr_16.', 'TestNr_17.', 'TestNr_18.', 'TestNr_19.']
sp_3 = ['TestNr_20.', 'TestNr_21.', 'TestNr_22.', 'TestNr_23.']
sp_4 = ['TestNr_24.', 'TestNr_25.', 'TestNr_26.', 'TestNr_27.']
sp_5 = ['TestNr_28.', 'TestNr_29.', 'TestNr_30.', 'TestNr_31.']
sp_6 = ['TestNr_32.', 'TestNr_33.', 'TestNr_34.', 'TestNr_35.']
specimen = [sp_1, sp_2, sp_3, sp_4, sp_5, sp_6]       # all measurement files
```

**Figure 15**: Example code

Additionally, within the developed Python script average values, such as the velocity of the hydraulic press or the furnace temperature, are calculated for each test setting. The Python script creates plots from the measured sensor data and saves them in a specified directory. In the following the measurement data of the sensors is outlined. For this purpose, measurements from experiment 2 are shown for each sensor.

The load cell converts the applied force of the hydraulic press into an electrical signal, which can be measured. Thereby, the electrical signal changes proportionally to the applied force [36]. Figure 16 shows the load-time curve for a test setting. At the beginning of the upsetting process a steep rise in the force of the hydraulic press can be seen. The curve levels off an then further increases nonlinearly until it reaches a maximum. As the top die moves up, force drops down to zero. Within the Python script, the average maximum force is determined for each test setting.

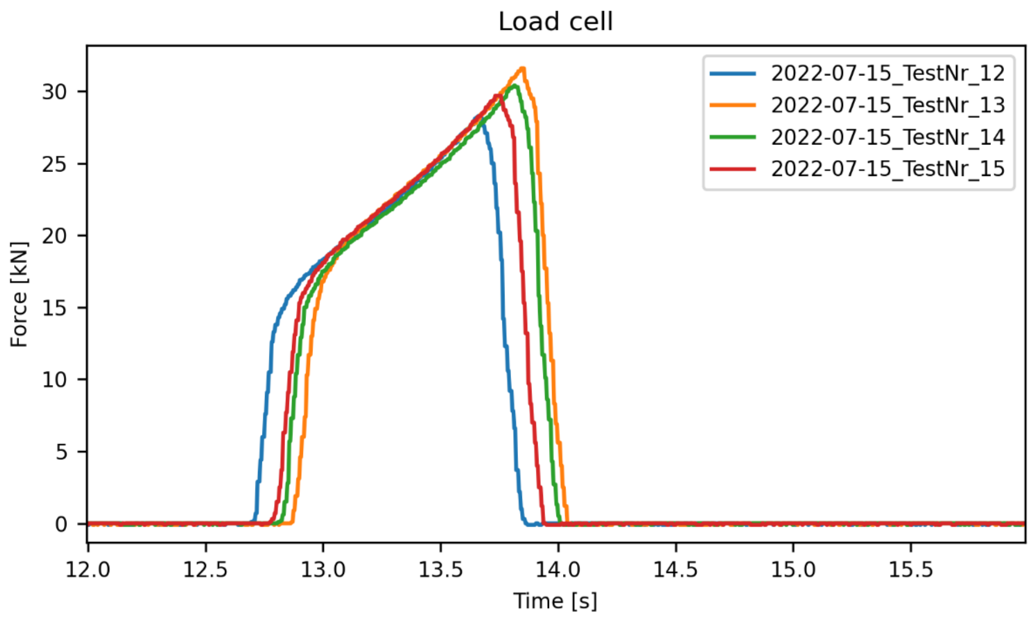


**Figure 16**: Sensor data - load cell

The LVDT converts linear movements into an electrical signal. Thereby, the position of the top die moving with constant velocity during the upsetting process, is determined, which is illustrated in Figure 17. The highest value indicates the top die being at the initial position, while the lowest value indicates the end of the upsetting process. Measurements show a slight deceleration of the top die at the time the die contacts the specimen. Upsetting velocity is calculated from measurement data of the LVDT, using the linear relation between the position of the top die and the time.

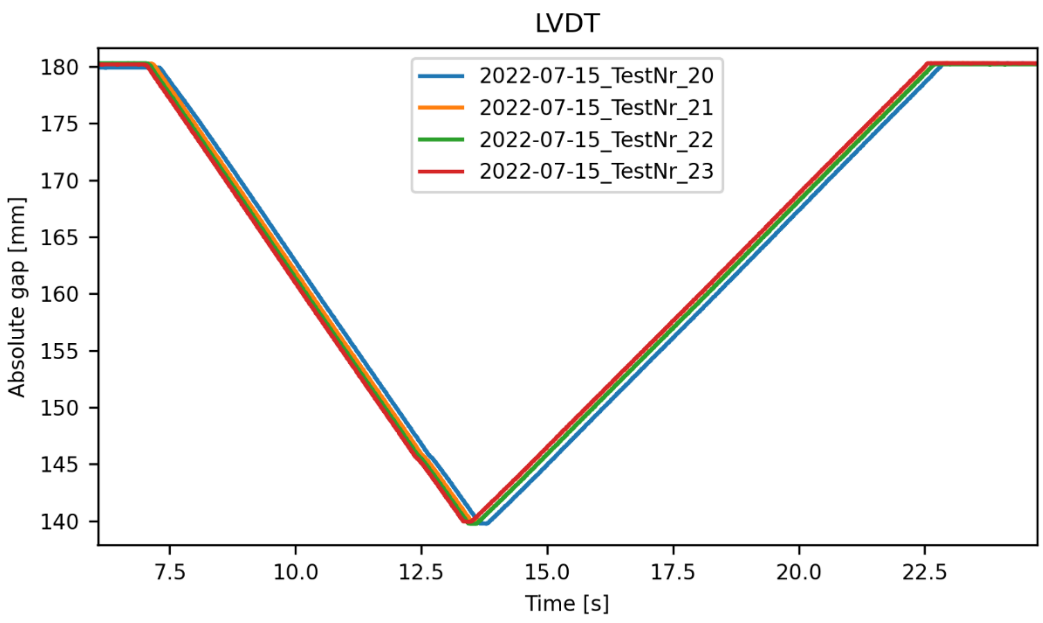


**Figure 17**: Sensor data - LVDT

$$v = \frac{\Delta s}{\Delta t} = \frac{s_{start} - s_{end}}{t_{end} - t_{start}} \tag{4.25}$$

The distance $\Delta s$ is calculated as the difference between the die position at the start of the die movement $s_{start}$ and the position at the end $s_{end}$ of the upsetting process. The time $\Delta t$ is calculated as the difference between the point of time the top die reaches the lowest position $t_{end}$ and the point of time at the beginning of the movement $t_{start}$. An average velocity for the measurements of the same setting is calculated for further use in the simulations.

A pyrometer is used for contactless temperature measurements. Radiation emitted by objects with temperatures greater than absolute zero temperature is detected from the pyrometer and transformed to an electrical signal [37]. In the experiment, the pyrometer measures the temperature of the specimen, shown in Figure 18.

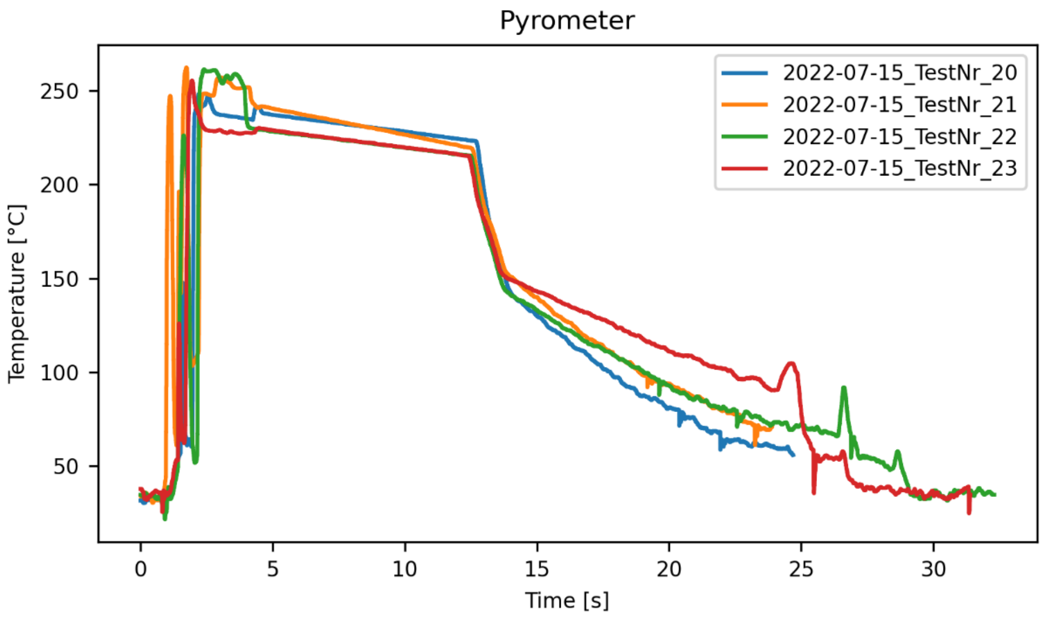


**Figure 18**: Sensor data - pyrometer

At the start of the measurement there is no specimen inserted into the hydraulic press, therefore the pyrometer indicates temperatures below 50 °C. At low temperature levels higher deviations between the measured temperature and the actual temperature occur. During positioning of the specimen on the bottom die, the temperature curve of the pyrometer shows high fluctuations as the specimen is manually moved by the gripper. Additionally, the pyrometer could measure the gripper temperature if it comes across the measuring position. In this case, the temperature curve shows significantly lower temperatures. As soon as the specimen is placed on the bottom die, the pyrometer shows a steady decrease in temperature. The temperature curve shows a steep drop during the upsetting process, as the specimen contacts both dies. A slight temperature rise due to deformation energy during upsetting can be seen in Figure 19. Thereby, temperature rise depends on the amount of the deformation. After the upsetting process, the specimen proceeds to cool down. Temperature decrease is higher, as the contact area between specimen and die has increased. As the specimen is removed, temperature immediately drops. In some cases, measurement stops before the specimen is removed.

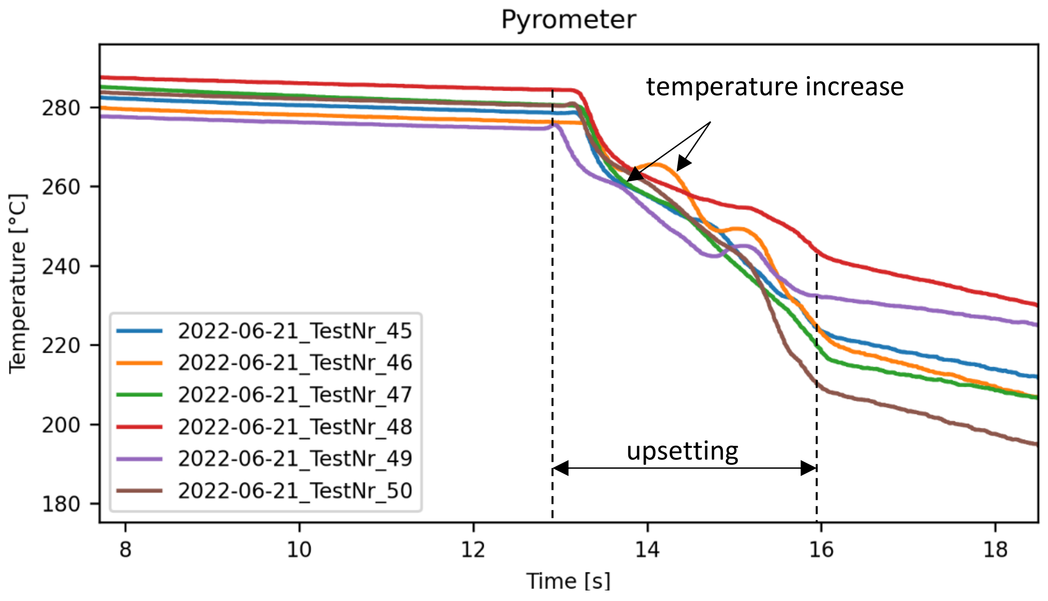


**Figure 19**: Sensor data - pyrometer (detail)

Thermocouples are simple and robust sensors to measure temperatures consisting of two different metals, that are joined together at one end. Due to heating or cooling of the junction, a voltage that correlates with the temperature is created [38]. The air temperature in the furnace is detected by the thermocouple installed via the retrofitting method. The temperature curve in Figure 20 indicates that the air temperature is not constant over the whole time. As the furnace is opened, hot air exchanges with the environment leading to a temperature loss inside the furnace. The average furnace temperature is calculated to assume the furnace temperature in the simulation. However, experiments showed that there are discrepancies between the measured temperature of the preinstalled thermocouples and the one that has been added with the retrofitting method, leading to uncertainties about the actual furnace temperature. Thereby, the retrofitted thermocouple, that indicates lower temperature values, is taken as reference.

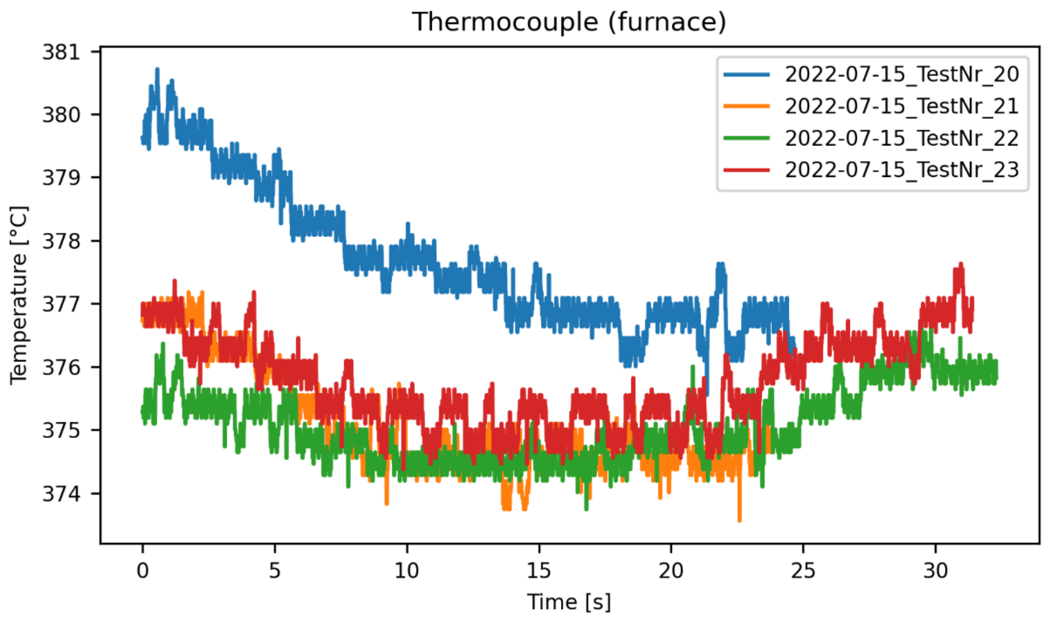


**Figure 20**: Sensor data - thermocouple furnace

Another thermocouple measures the temperature of the environment, see Figure 21, which is almost constant over the time. Also, an average temperature is calculated to estimate the environment temperature for the simulations.

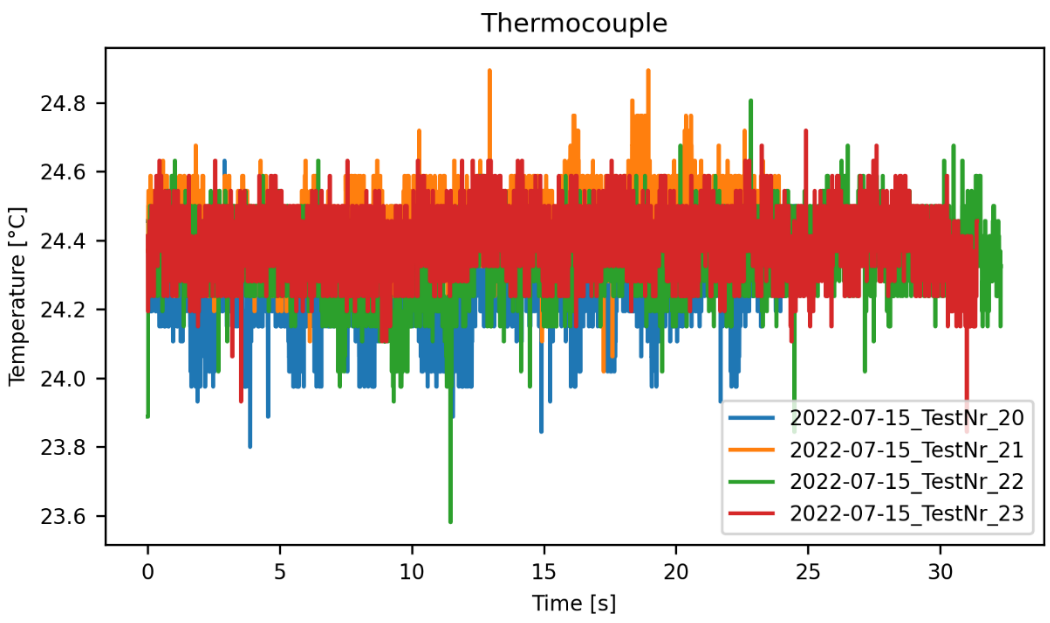


**Figure 21**: Sensor data - thermocouple

## 4.5. Data processing

To compare measurements and simulation, the sensor data is automatically processed using Python. Time-temperature curves and displacement-force curves were extracted from the sensor data. The temperature curve is determined for the timespan the specimen rests on the bottom die to the start of the upsetting process. Another temperature curve is determined for the time of the upsetting. Additionally, the force-displacement curve is calculated from the force of the load sensor and the top die position measured by the LVDT. Therefore, characteristic time points, illustrated in Figure 22, in the sensor data were identified. The time the specimen is placed on the bottom die $t_1$ is defined by the transfer time of the individual test setting, as it is very difficult to determine from the sensor data due to the fluctuations around this time point. The start of the upsetting process $t_2$ is defined by the force exceeding a defined threshold, whereas the end of the upsetting process $t_3$ is determined by the time the top die reaches the lowest position. Two temperature plots and the force displacement curve are created for each test setting. These plots serve as a basis to compare experiments and simulation.

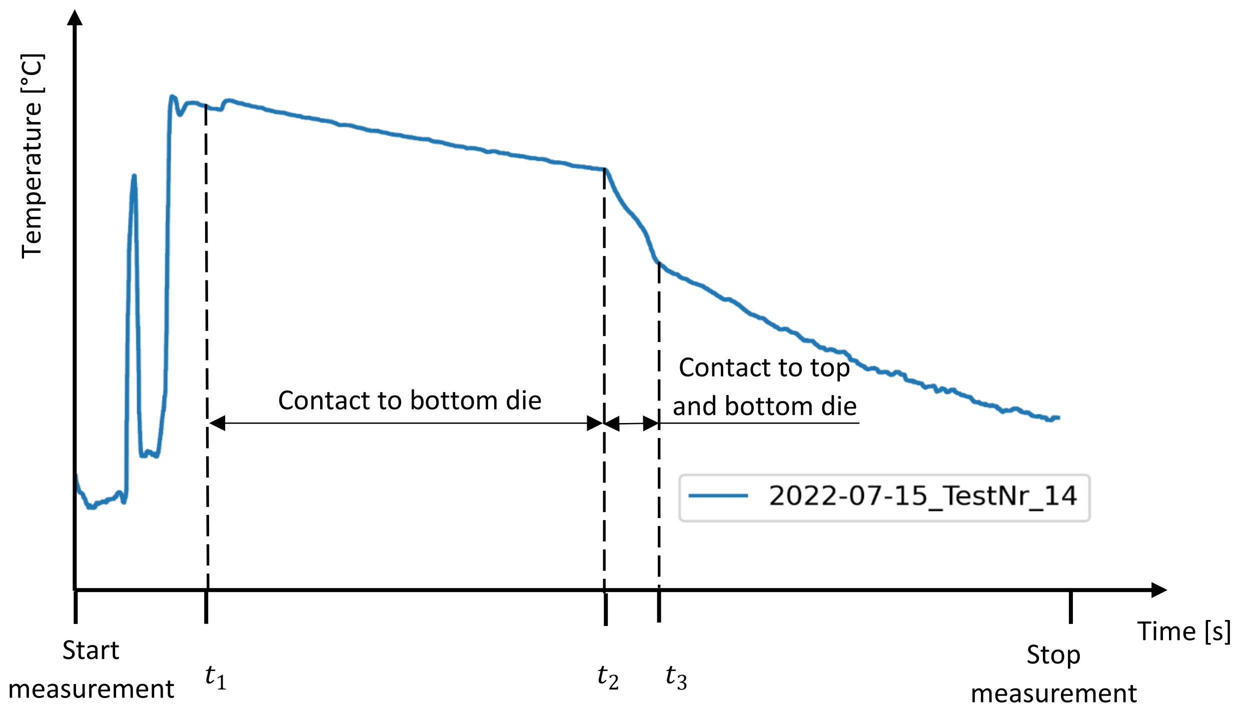


**Figure 22**: Relevant time points from the pyrometer measurement

# 5. Process simulation

This chapter deals with the simulation of the upsetting process cycle of the specimen. First, an overview is given on the process itself and how it is divided into individual simulations. Further, a focus is set on the development of each FE simulation to describe the whole process. The structure of the FE model, the analysis type, the mesh as well as the simulation outputs are briefly explained. Additionally, an alternative approach to calculate the temperature during the transport with Python is presented. Literature research on material properties and further relevant parameters is summarized and parameters for the Johnson-Cook material model used in the upsetting simulation are outlined.

## 5.1. Process cycle

Overall, the whole process is divided into four simulations, whereas three of them are transient heat transfer simulations describing the heating and cooling of the specimen and one is a fully coupled thermal-stress analysis to model the compression of the specimen during upsetting. The specimen is placed inside the preheated furnace for a defined heating time. Due to convection, conduction, and radiation the specimen temperature rises. This process step is obtained by Simulation 1. Subsequently, Simulation 2 represents the manual transport of the specimen from the furnace to the hydraulic press after the preheating. During the transport the specimen cools down due to radiation and convection to the environment. Additionally, heat conducts from the specimen to the gripper. Simulation 3 starts at the time the specimen is placed on the bottom die of the hydraulic press and ends at the time the top die has moved from the initial position to the top surface of the specimen. Meanwhile heat conducts from the specimen to the cooler die, additionally convection and radiation are present to a small extend. Finally, Simulation 4 includes the compression of the cylindrical specimen. During this process step, heat conducts from the specimen to both dies. Inside the specimen heat is generated, as energy that is expended to plastically deform materials is to a great extend converted into heat.

## 5.2. Unit system

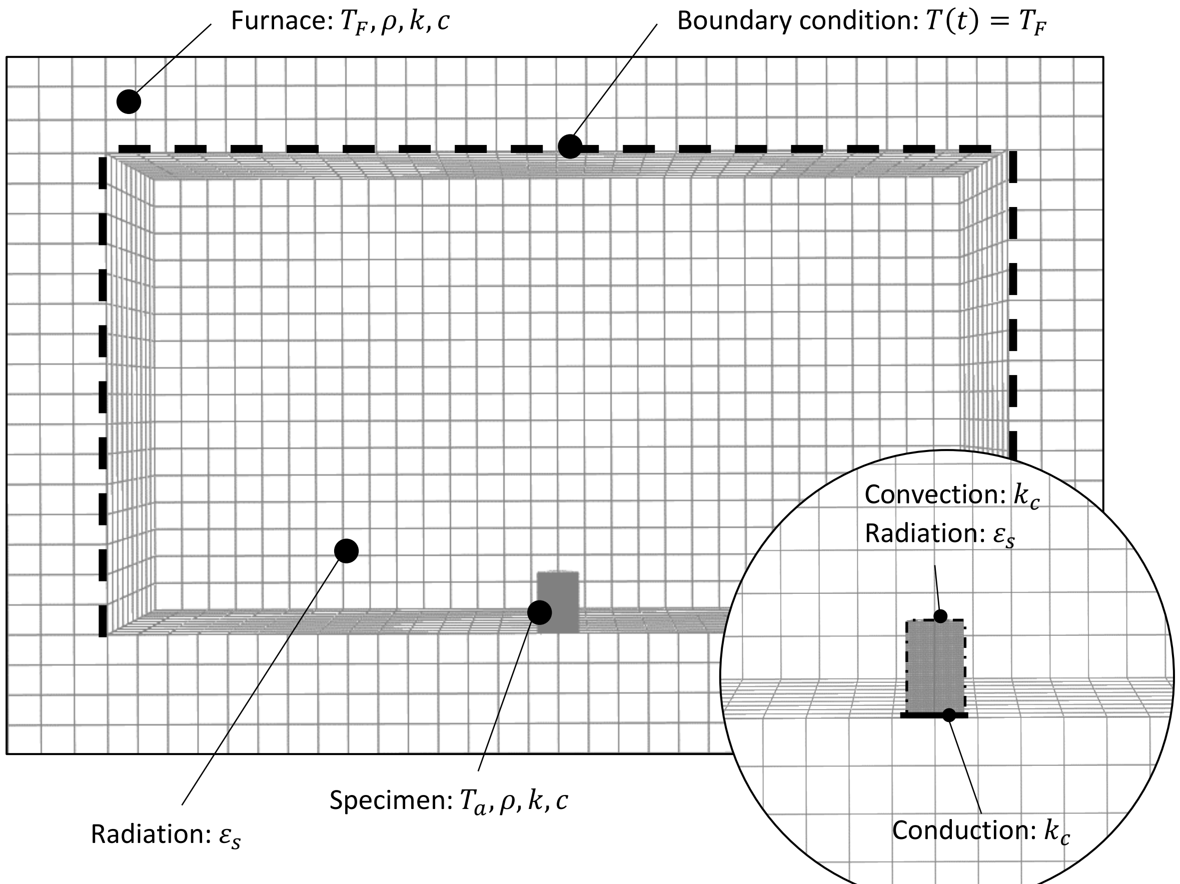
The FE simulations are carried out with Abaqus 2019. As there are no implemented units, all parameters are defined by using the SI-mm unit system, outlined in Table 9. Temperatures are defined in degree Celsius. Physical model parameters are defined for each FE model. The temperature at absolute zero is set at -273.15 °C and the Stefan Boltzmann constant $\sigma_k$ is defined as 5.67E-11 mW/mm$^2$°C$^4$ for consistency in units.

**Table 9:** Unit systems

| Quantity | SI | SI-mm |
|---|---|---|
| Length | m | mm |
| Force | N | N |
| Mass | kg | t ($10^3$ kg) |
| Time | S | s |
| Stress | Pa (N/m$^2$) | MPa (N/mm$^2$) |
| Energy | J | mJ ($10^{-3}$ J) |
| Density | kg/m$^3$ | t/mm$^3$ |

## 5.3. Simulation 1 - Heating

To simulate the process step of the preheating, furnace and specimen are modelled as three-dimensional parts. A schematic representation of the model, including material properties, interactions, initial and boundary conditions, is given in Figure 23.



Furnace: $T_F, \rho, k, c$
Boundary condition: $T(t) = T_F$
Convection: $k_c$
Radiation: $\varepsilon_s$
Specimen: $T_a, \rho, k, c$
Radiation: $\varepsilon_s$
Conduction: $k_c$

**Figure 23:** Simulation 1 - Heating

Material properties for specific heat, conductivity and density are defined for the specimen as well as for the furnace. It is assumed, that the furnace is already preheated. Therefore, the initial temperature of the furnace is set to this defined temperature, whereas the initial temperature of the specimen is defined equal to the ambient temperature. A surface-to-surface contact between the specimen and the furnace is defined. Furthermore, the thermal conductance in the contact area is specified to model the conductive heat transfer between furnace and specimen. Additionally, convection and radiation boundaries are applied. A transient 'heat transfer' step is applied, in which a boundary condition with constant temperature is defined on the inside walls of the furnace. The time period is defined by the heating time.

## 5.4. Simulation 2 - Transport

For the transport simulation an alternative approach is used instead of a FE simulation. A transient heat transfer equation was defined for the problem, which was solved using Python. Assuming that the temperature changes uniformly in the whole specimen, a differential equation for a 0-dimensional heat transfer problem is defined from equation (3.21). This energy balance equation considers temperature changes due to radiation and convection to the environment and a surface heat flux caused by the heat conduction from the specimen to the gripper.

$$\frac{dT}{dt} = \frac{1}{\rho V c}\left[q_s'' A_{s,h} - h_c\,(T - T_a)A_s - \varepsilon_s \sigma_k \left(T^4 - {T_a}^4\right) A_s\right] \tag{5.26}$$

$$\frac{T_{i+1} - T_i}{\Delta t} = \frac{1}{\rho V c}\left[q_s'' A_{s,h} - h_c\,(T_i - T_a)A_s - \varepsilon_s \sigma_k \left(T_i^4 - {T_a}^4\right) A_s\right] \tag{5.27}$$

$$T_{i+1} = T_i + \frac{\Delta t}{\rho V c}\left[q_s'' A_{s,h} - h_c\,(T_i - T_a)A_s - \varepsilon_s \sigma_k \left(T_i^4 - {T_a}^4\right) A_s\right] \tag{5.28}$$

The equation is discretized in time by applying the explicit Euler-method. Moreover, the equation is solved with Python with a defined number of iterations. As the initial condition, the temperature is taken from the previous simulation. The time delta $\Delta t$ is specified as 0.1 seconds since deviations to the calculated time with a delta of 0.01 are low. The number of iterations is calculated as the time of transport divided by the time difference. The term for the surface heat flux and the related surface area is unknown, therefore a correction term is used, and the temperature curve was fitted to the measurements.

## 5.5. Simulation 3 – Rest on die

The simulation model is illustrated in Figure 24. Bottom die and specimen are modelled as three-dimensional parts. Material properties for specific heat, conductivity and density are defined. The initial temperature of the bottom die is assumed to be at room temperature, whereas the initial temperature of the specimen is defined by the temperature at the end of the transport simulation. A surface-to-surface contact definition between specimen and die is created, whereas the contact conductance between the two parts is specified. Furthermore, convection to the environment is specified, whereas a heat transfer coefficient for free convection is considered. Although, radiation effects are rather small at lower temperatures and could be neglected, radiation to the environment is defined as the computation time is not high for this simulation.

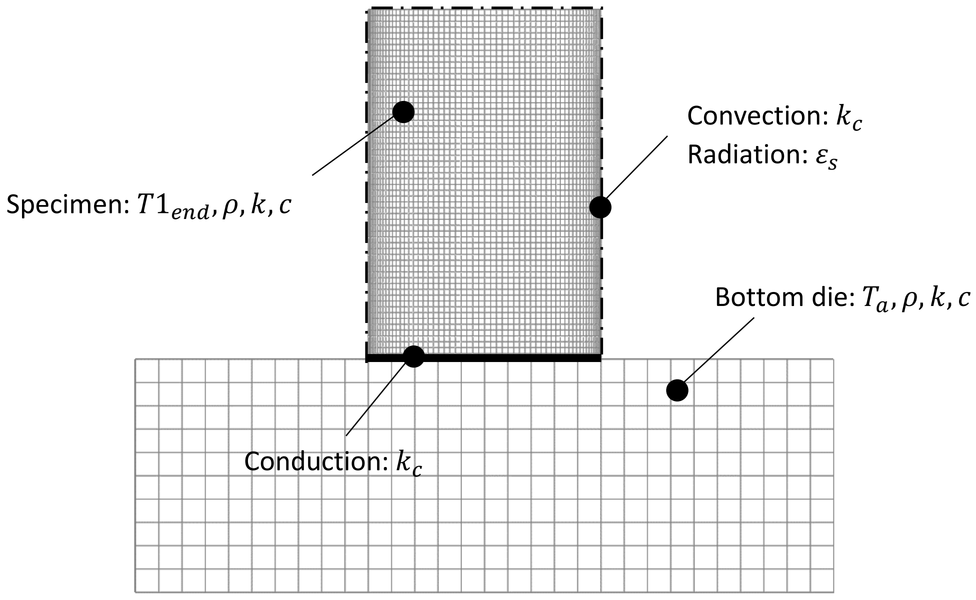

Convection: $k_c$
Radiation: $\varepsilon_s$
Specimen: $T1_{end}, \rho, k, c$
Bottom die: $T_a, \rho, k, c$
Conduction: $k_c$

**Figure 24**: Simulation 3 – Rest on die

## 5.6. Simulation 4 - Upsetting

For the upsetting simulation two different approaches, an explicit and an implicit, are elaborated. A schematic overview on the model, including material properties, initial condition, boundary conditions and interactions is given in Figure 25.Top and bottom die, as well as the specimen are modelled as three-dimensional parts. An initial gap of 0.1 mm between top die and specimen is defined to avoid problems with the contact definition. The initial temperature of the top die is defined equal to the ambient temperature, whereas the initial temperature of the specimen and the bottom die is defined by the temperature distribution at the end of the previous simulation. Contact between the parts is specified using a general contact formulation. Contact in normal direction is defined as 'Hard' contact, the tangential behavior is defined by the penalty friction formulation with a constant friction coefficient of $\mu = 0.3$. Heat conduction in the contact area to the dies is considered by defining the thermal contact conductance as a function of clearance. All degrees of freedom of the bottom die are constrained. A reference point is created and coupled to the contact surface of the top die. Thereby, all translational and rotational degrees of freedom are constrained. A displacement boundary is specified on the reference point by a time-displacement amplitude to define the movement of the top die. Heat loss due to convection and radiation is neglected, as the process time is rather short. Additionally, heat loss due to radiation is negligible for lower specimen temperatures. A 'Coupled temp-displacement' step is applied in the implicit simulation, for the explicit simulation the step is defined as 'Dynamic, temp-disp., Explicit'. Material properties for the dies and the specimen include specific heat, conductivity, density, and elasticity. Additionally, plastic material behavior is defined for the specimen material by using the Johnson-Cook constitutive equation. The explicit simulation includes damage for ductile materials. Heat generation due to plastic deformations are considered with the definition of the inelastic heat fraction.

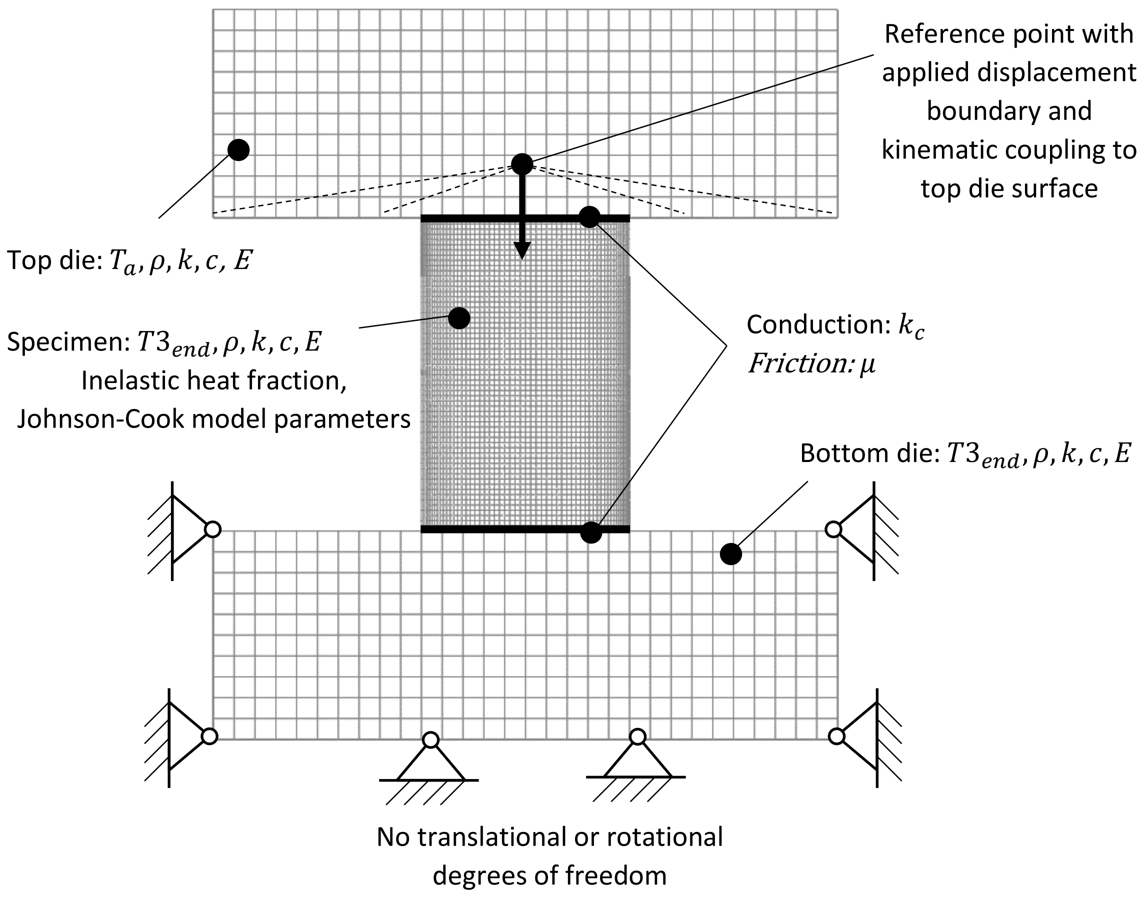

Reference point with applied displacement boundary and kinematic coupling to top die surface
Top die: $T_a, \rho, k, c, E$
Specimen: $T3_{end}, \rho, k, c, E$ Inelastic heat fraction, Johnson-Cook model parameters
Conduction: $k_c$
*Friction:* $\mu$
Bottom die: $T3_{end}, \rho, k, c, E$
No translational or rotational degrees of freedom

**Figure 25:** Simulation 4 - Upsetting

## 5.7. FE mesh

The specimen is partitioned by using datum planes, also mesh controls are applied to create a radial arrangement of the elements, illustrated in Figure 26. The element size is defined by applying global seeds. A hex-dominated mesh using sweep technique and the advancing front algorithm is created. Furnace and dies are meshed using hexagonal elements with the structured meshing technique. Additionally, partitions are created for the furnace.

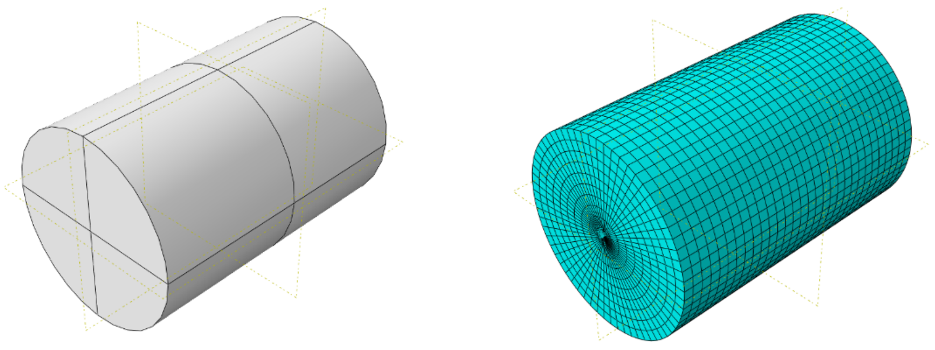


**Figure 26:** Partitions and mesh of the specimen

Depending on the type of simulation either elements for 'Heat transfer' or elements for 'Coupled Temperature-Displacement' are selected. The respective element library is used for implicit and explicit simulations. Element size is defined by global seeds for each part. Assigned element types are listed in Table 10 for each part in the Abaqus model. Thereby, element type DC3D8 is an 8-node linear heat transfer brick, DC3D6 is a 6-node linear heat transfer triangular prism. C3D8RT describes an 8-node thermally coupled brick with trilinear displacement and temperature, that uses reduced integration and hourglass control. C3D6T is a 6-node thermally coupled triangular prism with linear displacement and temperature [31].

**Table 10**: Element type for each part

| Part | Heat transfer | Coupled Temperature-Displacement |
|---|---|---|
| Specimen | DC3D8 + DC3D6 | C3D8RT + C3D6T |
| Dies | DC3D8 | C3D8RT |
| Furnace | DC3D8 | C3D8RT |

## 5.8. Simulation outputs

Field output and history output are requested in the simulation. The frequency of the field output is either defined as units of time depending on the step time or as numbers of intervals. Field output variables, such as nodal temperature, heat flux, stress, strain and so on are specified. Additionally, two different history outputs are defined. A node set, depicted in Figure 27, containing nodes on the shell surface of the cylindrical specimen at fixed x- and y- coordinate and variable z-coordinate is defined.

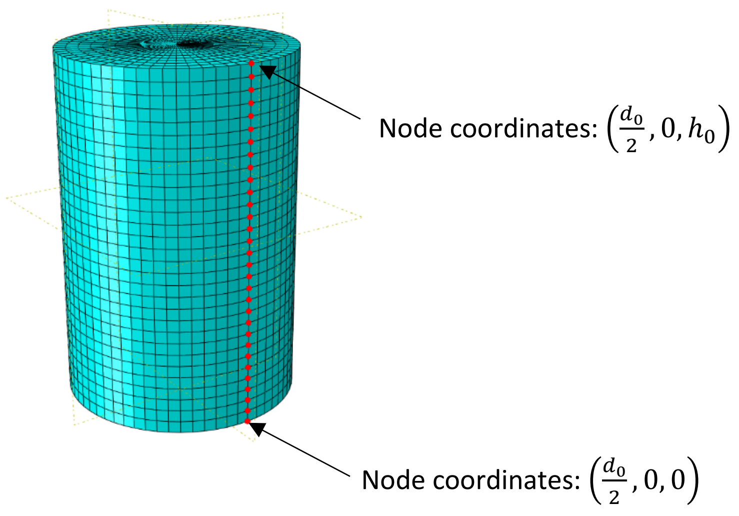

Node coordinates: $\left(\frac{d_0}{2}, 0, h_0\right)$
Node coordinates: $\left(\frac{d_0}{2}, 0, 0\right)$

**Figure 27:** Node set

History output, with nodal temperature 'NT' and nodal coordinate 'COORD' as output variables, is requested for this node set to evaluate the temperature of the specimen. Thereby, nodal coordinates are necessary to identify the nodes at the pyrometer position, which is further compared to the measured temperature. Another history output is created for the reference node in the upsetting simulation. The output variables are specified as U3, for the displacement along the z-axis, and RF3, which is the reaction force acting on the reference point, also along z-direction. The results of the history output are further accessed by a Python script to extract the results from the Abaqus output database.

## 5.9. Material properties

Material properties that need to be specified in a transient heat transfer analysis are the material density, specific heat, and conductivity. The thermal material properties of aluminum alloys, listed in Table 11, are assigned to the specimen. Thereby, the defined values for the specific heat capacity are for aluminum alloys in general and the conductivity values are for aluminum alloys in 6xxx series. The density of aluminum EN-AW 6082 is specified as $\rho = 2700\ kg/m^3$. This material parameters are used for the specimen in each simulation.

**Table 11**: Thermal material properties of aluminum alloys [39]

| Temperature $T$ [°C] | Conductivity $k$ [W/m°C] | Specific heat capacity $c$ [J/kg°C] |
|---|---|---|
| 20 | 191 | 911 |
| 100 | 197 | 944 |
| 200 | 204 | 985 |
| 300 | 211 | 1026 |
| 400 | 218 | 1067 |
| 500 | 225 | 1108 |

The furnace lining is made of refractory, therefore material properties of silica, a common refractory material, were selected. The thermal properties of silica, listed in Table 12, are assigned to the furnace in Simulation 1. The density of silica is defined as $\rho = 1820\ kg/m^3$ [40].

**Table 12**: Thermal material properties of silica [40]

| Temperature $T$ [°C] | Conductivity $k$ [W/m°C] | Specific heat capacity $c$ [J/kg°C] |
|---|---|---|
| 400 | 1.2 | 915 |
| 600 | 1.36 | 944 |
| 800 | 1.51 | 961 |
| 1000 | 1.64 | 969 |
| 1200 | 1.76 | 979 |

The dies of the hydraulic press are probably made from hot-working steel, e.g., W300. However, as the exact material specification is unknown, material properties of carbon steel are assumed, as temperature dependent properties were found in the literature. The density is specified as $\rho = 7850\ kg/m^3$, thermal material properties for steel used in the simulation are shown in Table 13.

**Table 13**: Thermal material properties of carbon steel [41]

| Temperature $T$ [°C] | Conductivity $k$ [W/m°C] | Specific heat capacity $c$ [J/kg°C] |
|---|---|---|
| 20 | 53 | 440 |
| 100 | 51 | 488 |
| 200 | 47 | 530 |
| 300 | 44 | 565 |
| 400 | 41 | 606 |
| 500 | 37 | 667 |

Elastic behavior is described by a linear isotropic elasticity model, characterized by the Young's Modulus and the Poisson's ratio. The temperature dependent Young's modulus for aluminum is shown in Table 14, the Poisson ratio of aluminum is assumed to be 0.33 [21].

**Table 14**: Young's modulus for aluminum alloys [39]

| **Temperature $T$ [°C]** | **Young's modulus $E$ [MPa]** |
|---|---|
| 20 | 70000 |
| 50 | 69300 |
| 100 | 67900 |
| 150 | 65100 |
| 200 | 60200 |
| 250 | 54600 |
| 300 | 47600 |
| 350 | 37800 |
| 400 | 28000 |

The type of simulation requires elastic behavior to be specified for all parts in the simulation. The Young's modulus and the Poisson ratio for steel are listed in Table 15.

**Table 15**: Young's modulus and Poisson ratio of steel [42]

| **Temperature $T$ [°C]** | **Young's modulus $E$ [MPa]** | **Poisson ratio $\nu$ [-]** |
|---|---|---|
| 50 | 206400 | 0.271 |
| 100 | 201600 | 0.271 |
| 150 | 198300 | 0.273 |
| 200 | 193300 | 0.275 |
| 250 | 190600 | 0.278 |
| 295 | 186400 | 0.282 |

Viscoplastic material behavior is defined with the Johnson-Cook constitutive material model, that describes the behavior of metals considering work hardening in the first term, strain rate hardening in the second term and thermal softening of the material in the third term [43].

$$\sigma = \left(A + B\,\varepsilon_p{}^n\right)\left[1 + C\ln\left(\frac{\dot{\varepsilon}_p}{\dot{\varepsilon}_0}\right)\right]\left[1 - \left(\frac{T - T_t}{T_m - T_t}\right)^m\right] \tag{5.29}$$

In this equation σ is the stress, $A$ the yield strength of the quasi-static condition, $B$ the strain hardening constant, $\varepsilon_p$ the plastic strain, $n$ the strain hardening exponent, $C$ the strain rate sensitivity, $\dot{\varepsilon}_p$ the strain rate, $\dot{\varepsilon}_0$ the reference strain rate, $T$ environment temperature, $T_t$ the reference temperature, $T_m$ the melting temperature [43]. The Johnson-Cook plasticity model can be described as particular type of isotropic material hardening. The Johnson-Cook material model can be used together with the Johnson-Cook dynamic failure model enabling to evaluate material failure. Damage of the material occurs if the damage parameter $\omega$, which is defined as [31]

$$\omega = \sum \left( \frac{\Delta\varepsilon_{pl}}{\varepsilon_{f\,pl}} \right) \tag{5.30}$$

exceeds 1, whereby $\Delta\varepsilon_{pl}$ is an increment of the equivalent plastic strain and $\varepsilon_{f\,pl}$ describes the strain at failure. The failure model describes the strain at failure dependent on a nondimensional plastic strain rate $\frac{\dot{\varepsilon}_p}{\dot{\varepsilon}_0}$, the ratio of the pressure stress to the Mises stress $\frac{p}{\sigma}$ and the nondimensional temperature $\frac{T-T_t}{T_m-T_t}$, which is also defined in the Johnson-Cook plasticity model. In this equation $D_1 - D_5$ are the failure parameters [31].

$$\varepsilon_{f\,pl} = \left[ D_1 + D_2\, exp\left( D_3 \cdot \frac{p}{\sigma} \right) \right] \left[ 1 + D_4 \ln\left( \frac{\dot{\varepsilon}_p}{\dot{\varepsilon}_0} \right) \right] \left[ 1 + D_5 \cdot \left( \frac{T - T_t}{T_m - T_t} \right) \right] \tag{5.31}$$

Various parameters for the Johnson-Cook material model for aluminum EN AW-6082 can be found in the literature: [43–50]. Some of the parameters for the Johnson-Cook model, obtained from literature, are shown in Table 16. Additionally, damage parameters are listed in Table 17.

**Table 16:** Johnson Cook Model parameters

| $A$ | $B$ | $C$ | $n$ | $m$ | $\dot{\varepsilon}_0$ | $T_m$ | $T_t$ | Literature |
|---|---|---|---|---|---|---|---|---|
| [MPa] | [MPa] | [-] | [-] | [-] | [s$^{-1}$] | [°C] | [°C] | |
| 201.55 | 250.87 | 0.00977 | 0.206 | 1.31 | 0.001 | 582 | 20 | [44] |
| 297.8 | 111.1 | 0.0238 | 0.048 | 1.19 | 1 | 555 | 25 | [45] |
| 285 | 94 | 0.002 | 0.41 | 1.34 | 1 | 588 | 25 | [47] |
| 250 | 243 | 0.00747 | 0.17 | 1.31 | 1 | 582 | 25 | [49] |

T**able 17**: Johnson Cook damage parameters [44]

| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|
| [-] | [-] | [-] | [-] | [-] |
| 0.0164 | 2.245 | -2.798 | 0.007 | 3.65 |

## 5.10. Temperature increase associated with plastic deformations

In metal forming energy is expended to plastically deform materials, whereby this energy is to a large extend converted into heat, leading to an increase in the component temperature [51]. In the work of [50], experiments are conducted to measure the temperature increase of an aluminum alloy EN AW-6082 via infrared thermography, which is a commonly used technique. The medium value for the fraction of plastic deformation that is converted into heat was calculated as 0.9 within this work. This value is also a very commonly used value for metals in general [46, 50]. In further literature [44], the fraction of plastic work, that is converted into heat is defined as 0.9 for aluminum 6082, although, literature shows, that this value depends on the strain rate.

Abaqus provides the possibility to include the heat generation by defining an inelastic heat fraction, to specify the fraction of inelastic dissipation applied as heat flux per volume. The inelastic heat fraction can be defined in conjunction with the Johnson-Cook plasticity model, the density and the specific heat [31]. Bulk metal forming processes, for example, involve large amounts of inelastic strain. Considering the heat generation allows for a more realistic process simulation, as material properties depend on temperature. For the present work, the inelastic heat fraction was defined as 0.9.

## 5.11. Thermal contact conductance

The heat transfer from a component to the tools has various impacts on the process, for example during forging. The thermal contact conductance depends on several parameters, but most importantly on the geometry of the contact surfaces, the contacting materials, the pressure, the temperature and the lubrication type [51].

Heat is transferred by conduction through contact asperities. Pressure and surface roughness define the asperity shape. High contact pressure leads to deformation of the asperities, which increases the contact area and the heat transfer coefficient as well. Furthermore, decreasing surface roughness leads to higher conductance in the contact area [52]. Contact conductance can be measured from experiments. Another method is to vary the parameter in numerical solutions to adapt the results to a measured temperature distribution. The contact conductance gives the best match between experiment and simulation [51]. Literature provides reference values for the thermal contact conductance, see Table 18.

**Table 18**: Reference values - contact conductance

| **Application** | **Contact conductance [W/m$^2$K]** |
|---|---|
| Hot pressing of aluminum [51] | 15000 … 30000 |
| Approximate value for hot forming [51] | 50000 |
| Aluminum during hot forming [53] | 3300 |
| Aluminum – aluminum [54] | 2200 - 12000 |
| Stainless steel – stainless steel [54] | 2000 - 3700 |
| Ti-6Al-V4 workpiece - H13 steel die [55] | 4000 – 6000 |

In Abaqus the contact conductance is defined as a function of clearance and / or a function of pressure. In the simulations the contact conductance is defined dependent on the gap between the contact surfaces. For the upsetting simulation, a higher heat transfer coefficient is used than in the heat transfer simulation to the die. Thereby, the contact conductance is defined based on reference values and further parameter variation in order to fit the simulation to the measurements.

## 5.12. Convection coefficient

Literature, given in Table 19, provides reverence values for the heat transfer coefficient used in calculations with forced or natural convection. Thereby, higher values indicate higher heat loss or heat input.

**Table 19**: Reverence values - convective heat transfer coefficient

| **Application** | **Heat transfer coefficient $h_c$ [W/m$^2$K]** |
|---|---|
| Free convection [56] | 3 … 20 |
| Forced convection [56] | 10 … 100 |
| Forced convection [22] | 25 … 250 |

# 6. Abaqus Scripting

As multiple simulations with varying geometry and input parameters are required, the generation of the Abaqus models as well as the evaluation of the simulation results is automized by using Abaqus-specific Python commands. This chapter gives an overview on the Abaqus Scripting Interface and object-oriented programming. The hierarchy of the Abaqus output database is illustrated to demonstrate how to access data of an Abaqus output database. Additionally, the structure of the developed Python code is outlined.

## 6.1. Abaqus Scripting Interface

The Abaqus Scripting Interface is an application programming interface (API) that extends the object-oriented programming language Python. From a script, containing Abaqus Scripting Interface commands, Abaqus/CAE functionalities can be accessed. For instance, the Abaqus Scripting Interface

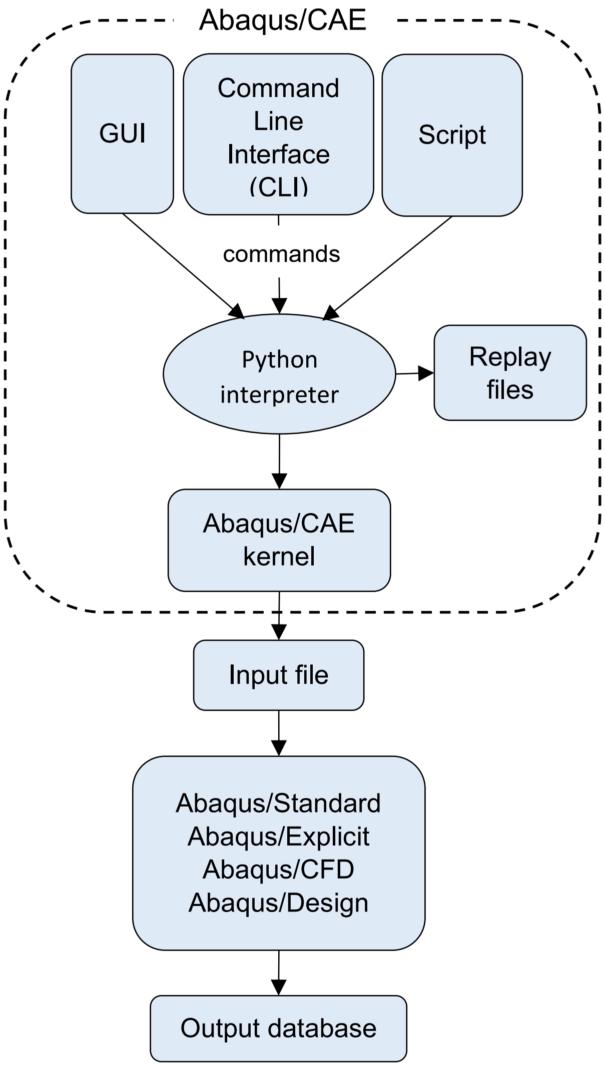


**Figure 28**: Interaction of Abaqus Scripting Interface commands with the Abaqus/CAE kernel [31]

allows the user to create and modify an Abaqus model, submit jobs, read from an output database, or view analysis results. Figure 28 depicts the interaction of Abaqus Scripting Interface commands with the Abaqus/CAE kernel. The Abaqus/CAE graphical user interface (GUI) allows the user to interact with the kernel. It generates Python commands based on the selected options and settings from dialog boxes, which are then interpreted by the Abaqus/CAE kernel. All commands are stored in the replay (.rpy) file. Instead of the Abaqus GUI, a script that contains Abaqus Scripting Interface commands, can be used to directly communicate with the kernel. Additionally, a script allows the automation of repetitive tasks [31].

## 6.2. Recording Python commands

A detailed introduction on recording Python commands from Abaqus/CAE to create a script is given in [57], which mentions the following options to record the commands: (1) Each click in the Abaqus GUI, even scrolling or zooming in the Abaqus Viewer, is recorded and the Python commands are automatically saved into the replay (.rpy) file in the active directory. (2) As the Abaqus model is saved, additionally, a journal (.jnl) file is saved. This includes only commands necessary for the model generation. (3) Also, the Macro Manager can be used to record and write commands to the abaqusMacros.py file until the recording is stopped. Recorded commands can be used to develop the Python script [57]. Further information can also be found in the Abaqus Scripting reference [31].

In this work, the recorded Python commands from the replay file were used to build functions. To create adaptable simulations, parameters were used when necessary.

## 6.3. Object-oriented programming

Python is an object-oriented programming language, which means it is based around objects. Objects include data, referred to as the member of an object. So called methods are used to manipulate the data of an object. An example for a Python object could be the model of a real-world object, like a tire, or even an array of nodes. In case of the tire, the encapsulated data could be its width, diameter, or the price. Methods, for instance, calculate deformation or wear of the tire during use. Different types of objects can share the same members and methods. Furthermore, class definitions include members and methods operating on the members [31].

## 6.4. Abaqus Output Database (ODB)

Abaqus saves results data and model data in an output database. Field outputs as well as history outputs defined in the Abaqus model are stored in the results data and can be accessed by Abaqus Scripting. An ODB-object is created if an output database is opened. Each step is defined as a member of the ODB-object. Further, the step-object contains field outputs and history outputs. To access field or history outputs, Python commands are used to step through the hierarchy, shown in Figure 29 [31]. For instance, to access the reaction force acting on a reference point, the following structure can be used:

```
odb.steps[‘stepname’].historyRegions[‘regionname’].historyOutputs[‘variable’].data
```

Thereby, odb is the created output database object, ‘stepname’ is the name defined for the step for which the data should be evaluated, ‘regionname’ is the name of the history region, which is defined by Abaqus, and ‘variable’ is the desired output variable – in this case ‘RF’.

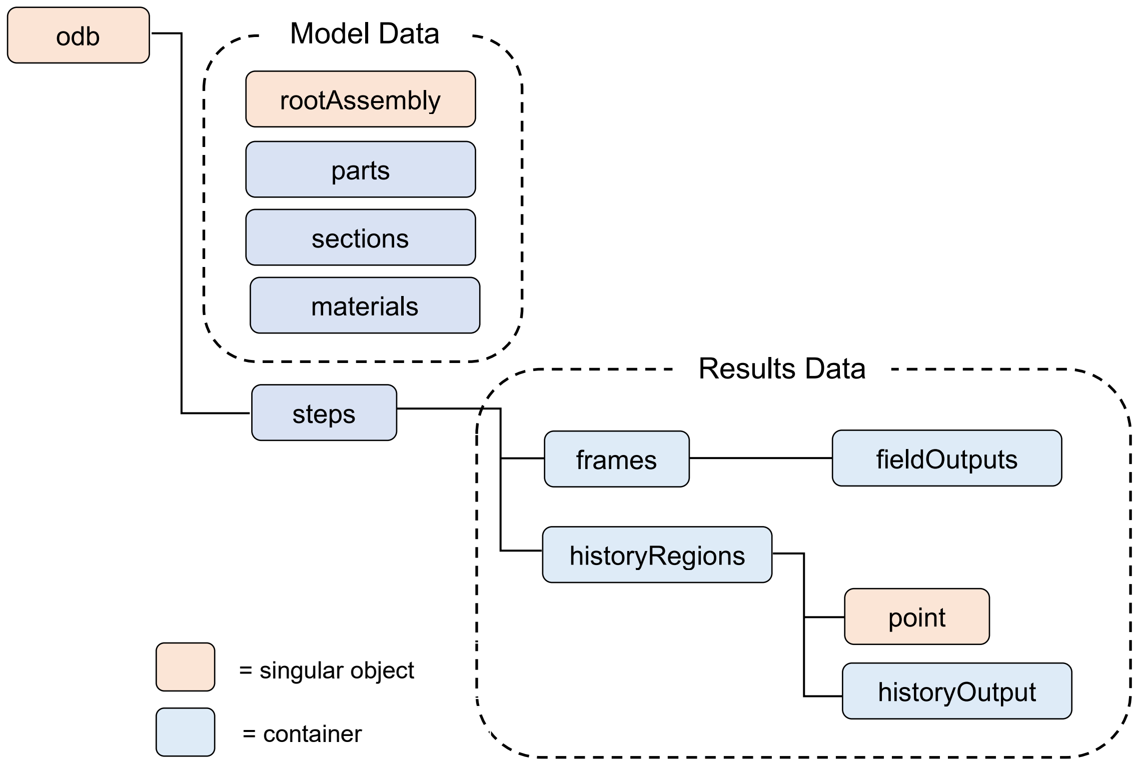


**Figure 29**: Abaqus Output Database [31]

## 6.5. Script structure

For editing the Python scripts, the PyCharm Community Edition 2021.2.3 was used. In the following, an overview on the functionalities and the file structure, illustrated in Figure 30, is given. For reasons of clarity and simplicity the Abaqus Scripting Interface commands are separated into a main script and modules. Thereby, each module contains a set of functions to access Abaqus/CAE functionalities. Further, modules are divided into layer 1 and layer 2 modules. Layer 1 modules are directly called from the main script, whereas layer 2 modules are called in layer 1 modules. Therefore, all layer 1 modules can use the same functions, defined in layer 2 modules. Changes in a layer 2 module, such as material properties, are adapted for all layer 1 modules that access this type of information.

The main script includes the necessary Abaqus specific import statements and import statements that include the modules. In the first section, the file paths - one for the Abaqus results and one for the .csv files - are specified. Subsequently, the specimen geometry as well as process parameters, such as furnace temperature, transport time or the velocity of the hydraulic press, are defined. Additionally, simulation parameters, like simulation name, friction coefficient, emissivity, or element size need to be specified. Defined file names serve for identification purpose, as they are also used in other scripts. The main script starts the simulations, evaluates the output database and writes the simulation results to a .csv file in the specified directory.

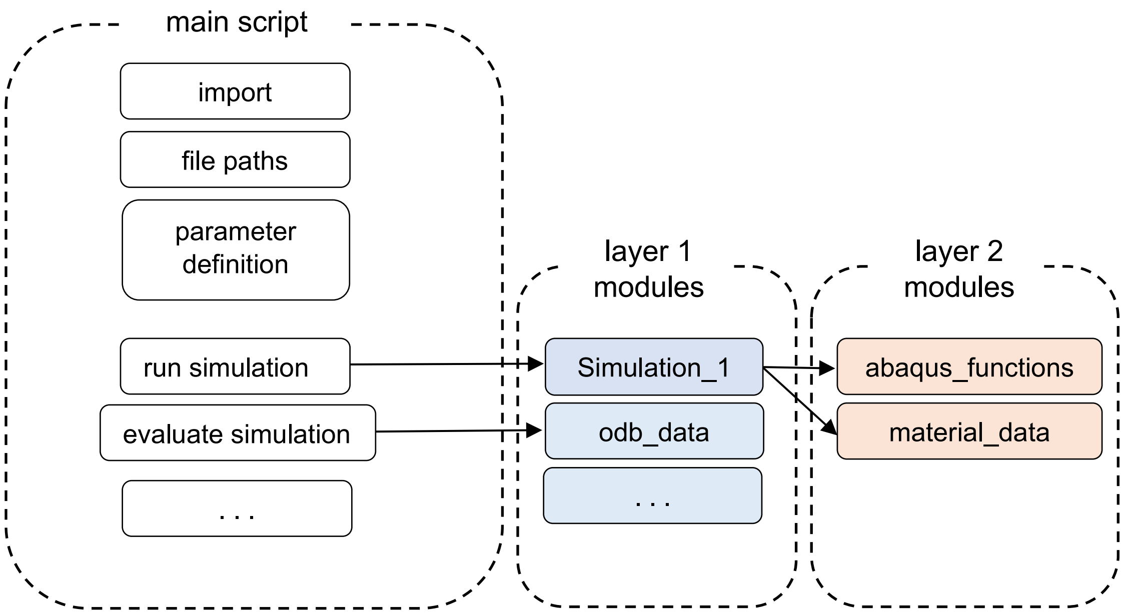


**Figure 30**: Python script structure

The main script offers the possibility to choose, which simulation to run. Although, if, for example, only the last simulation is executed, an .odb file of the previous simulation needs to be present in the specified directory. The main script is executed via the command window, either by starting the Abaqus GUI or without GUI:

```
abaqus cae script=main_script.py
abaqus cae noGUI=main_script.py
```

Also, the script can be started using the GUI -> File -> Run Script. Before executing the script, the working directory needs to be specified, which needs to be the directory of the main script. Otherwise

Abaqus has no access to the modules. If changes are made in one of the modules, Abaqus needs to reload the module again, therefore Abaqus needs to restart.

Layer 1 modules are all modules that either create and run a simulation or evaluate the results of a simulation. All 'Simulation' modules contain one function, that generates and runs the simulation. Thereby, all modules, except the one used for the Python heat transfer simulation, use the functions defined in the 'abaqus_function' module to build the Abaqus model step by step and run the simulation. Changes in the structure of the simulation models can be achieved by editing the respective 'Simulation' module. General input parameters for these modules are, for instance, the specimen diameter and height, process time, mesh size and so forth.

In the simulations, the nodal temperatures for a defined node set and the reaction force and displacement of the reference node are defined as history output and therefore, the corresponding values are saved in the output database. To make the simulation results available for the comparison between experiment and simulation, results are extracted from the Abaqus .odb file and saved as a .csv file by using the 'odb_data' module. Therefore, some general functions are defined, which are used in three different evaluation functions. The first evaluation function saves the nodal temperatures for each node defined in the node set to a .csv file. The first column contains the z-coordinate of the node, the second the time and the third one the nodal temperature. The second evaluation function calculates an average temperature, using the temperatures of the nodes at the end of a step. The third evaluation function extracts the force and displacement of the reference node. All output variables are saved into a .csv file. Thereby, in the first column the time is specified, in the second one the displacement and in the third one the reaction force.

There are only two modules, named 'material_data' and 'abaqus_functions', that belong to layer 2. The first one mentioned includes the material properties of aluminum EN AW-6082, steel, and silica, providing the benefit that the same material properties are used in each simulation. If material properties change, it only needs to be adapted in this module. Literature sources for the material properties are mentioned in chapter 5.9. The second one mentioned is the basis for the scripting of the FE models with Abaqus. This module contains general functions, that require input parameters to execute desired commands in Abaqus/CAE. For example, this module includes a function that creates a part in Abaqus with defined dimensions, a function to generate the mesh of a part and many more to build the FE model step by step. For further details, short descriptions of the functions are provided in each script in the appendix.

# 7. Concept and Implementation

In this chapter, an overview is given on the whole process, including the visualization of the sensor data, the automized simulation and the comparison between experiment and simulation. Furthermore, the automation of the simulation process is outlined in detail. Illustrations are presented to describe the workflow.

## 7.1. Overall process

The overall process, illustrated in Figure 31, is realized by using Python. First, an experimental plan is necessary to define the process parameters for each test setting. After the experiments are conducted, sensor data is available. To use the 'measurements.py' script (Appendix A), information about the test setting needs to be defined manually in the Python script. To visualize the measured sensor data, the script accesses all measurement files in a specified folder. As an output, the measured quantities are represented over the time in a diagram. All measurements belonging to the same test setting are

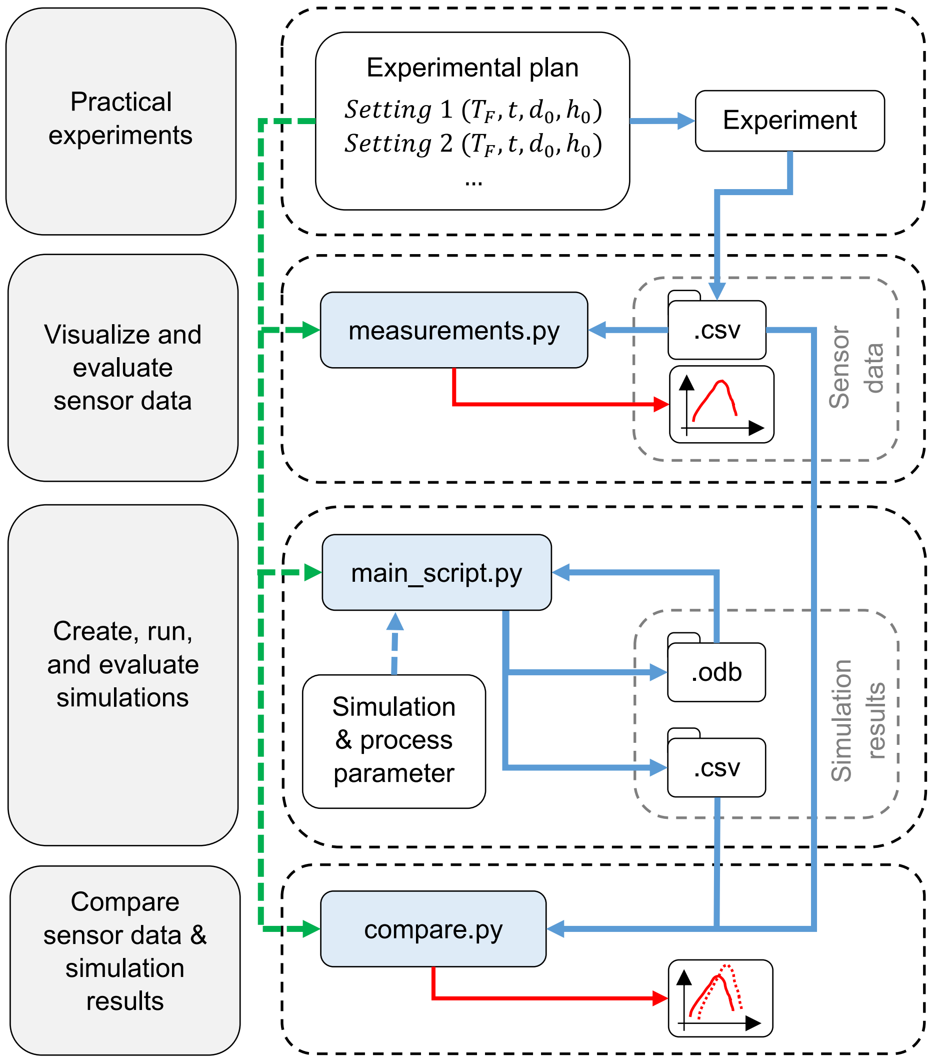

Practical experiments
Experimental plan
$Setting\ 1\ (T_F, t, d_0, h_0)$
$Setting\ 2\ (T_F, t, d_0, h_0)$
...
Experiment
Visualize and evaluate sensor data
measurements.py
.csv
Sensor data
Create, run, and evaluate simulations
main_script.py
.odb
.csv
Simulation & process parameter
Simulation results
Compare sensor data & simulation results
compare.py

**Figure 31**: Overall process

illustrated in the same diagram. As a result, the visualized sensor data can be checked for plausibility. The ‘main_script.py’ (Appendix B) script is responsible for creating, running, and evaluating the simulations. Manual input is needed to define process and simulation parameter. The simulation part can also be executed before the experiments. If measurements already exist, some parameters, such as the furnace temperature, can be estimated from the ‘measurements.py’ script. In ‘main_script.py’, file paths for the .odb files and the .csv files need to be specified. This script will be further explained in the next section. Finally, the ‘compare.py’ script combines the measurements and the simulation results. Therefore, the sensor data needs to be processed to make it comparable to the simulation results. As in the ‘measurements.py’ script, information about the test settings needs to be specified at the beginning of the script. Further, the folder paths need to be specified.

## 7.2. Automation of the simulation sequence

By using the ‘main_script.py’ the simulation process is executed, as depicted in Figure 32. Required input parameters are file paths for the results, process parameters and simulation parameters. The name of the simulated setting needs to be specified, as defined in the experimental plan, to assign the simulation to the corresponding measurement later. The names for the simulations are already defined as they are used as keywords in the ‘comparison.py’ script. In this main script, the corresponding modules are called in the right order.

The first module is the ‘Simulation_1.py’ (Appendix D), which represents the heating of the specimen. Results of the FE simulation are saved in a specified folder. The module ‘odb_data.py’ (Appendix I) evaluates the Abaqus output database saved in the specified directory and determines the average temperature at the end of the heating simulation. This temperature serves as initial temperature for the transport simulation implemented with Python, named ‘Simulation_2p.py’ (Appendix E). The resulting temperature is then again used as initial temperature for the next module ‘Simulation_3p.py’ (Appendix F), which represents the heat transfer to the bottom die before the compression of the specimen. The FE simulation provides an output database, which is evaluated by using the ‘odb_data.py’ module. Relevant data from the output database is saved in a .csv file. The temperature field at the end of this simulation is further used to define the initial temperature in the upsetting simulation, represented as ‘Simulation_4i.py’ (Appendix G) for the implicit simulation, or ‘Simulation_4e.py’ (Appendix H) for the explicit simulation. The ‘main_script.py’ offers the possibility to choose between the implicit and explicit simulation. As a result, again an output database is generated, which is accessed by the ‘odb_data.py’ module to create .csv files for further comparison between simulation and experiment. Thereby, all ‘Simulation_XX.py’ modules access the material properties defined in the ‘material_data.py’ (Appendix J) module. Additionally, all FE simulations access the 'abaqus_functions.py' (Appendix K) module, which defines functions to generate and run an Abaqus simulation. Additionally, if intermediate results need to be examined, there is the option to run only simulation 1, or simulation 1-3. If results of the previous simulations are already stored in the directory, it is possible to execute only simulation 4.

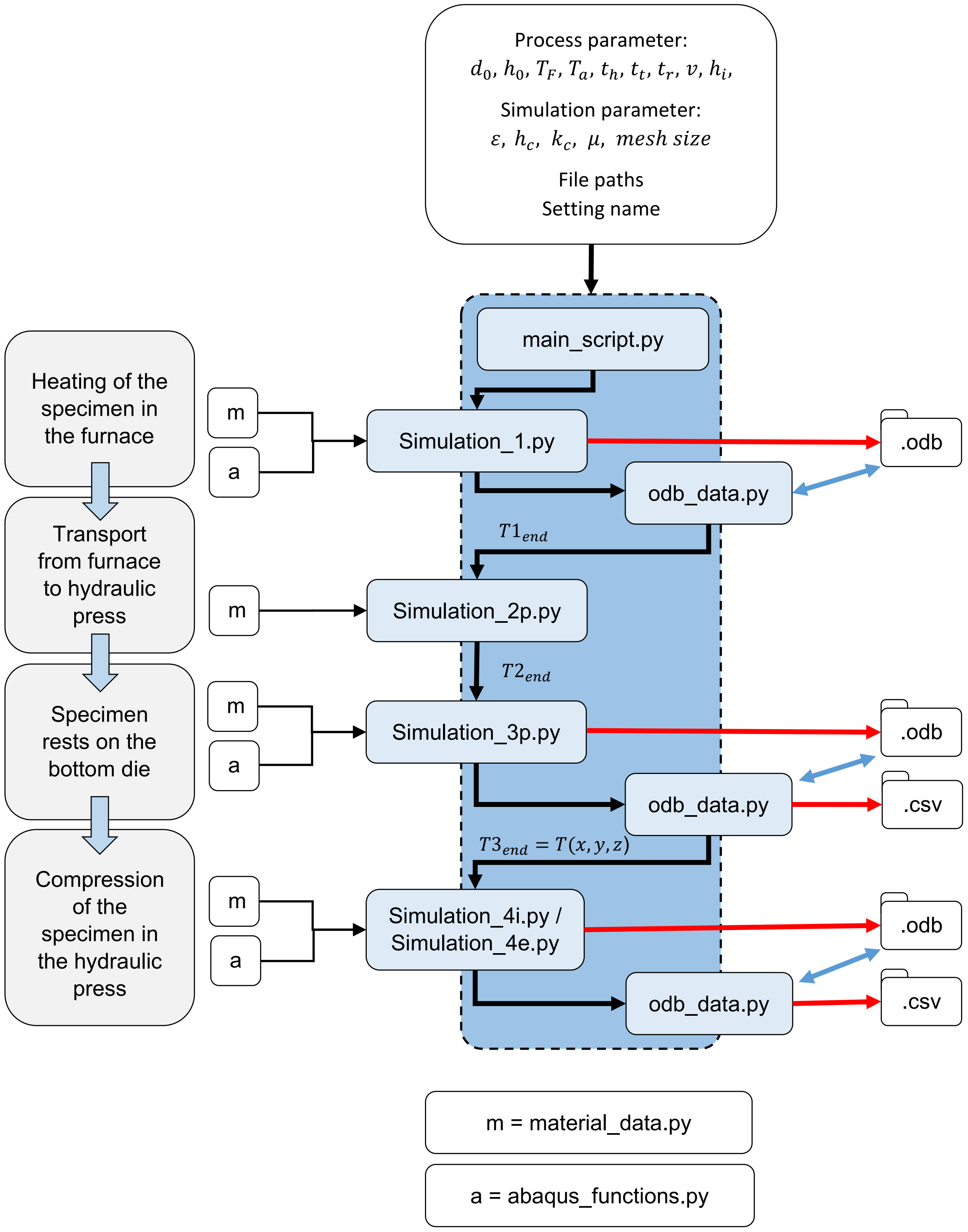

Process parameter:
$d_0, h_0, T_F, T_a, t_h, t_t, t_r, v, h_i,$
Simulation parameter:
$\varepsilon,\ h_c,\ k_c,\ \mu,\ mesh\ size$
File paths
Setting name
main_script.py
Heating of the specimen in the furnace
Transport from furnace to hydraulic press
Specimen rests on the bottom die
Compression of the specimen in the hydraulic press
m
a
Simulation_1.py
.odb
odb_data.py
$T1_{end}$
Simulation_2p.py
$T2_{end}$
Simulation_3p.py
.odb
odb_data.py
.csv
$T3_{end} = T(x,y,z)$
Simulation_4i.py / Simulation_4e.py
.odb
odb_data.py
.csv
m = material_data.py
a = abaqus_functions.py

**Figure 32:** Automation of the simulation process

**Table 20:** Input and output parameter of each simulation

| **Input Simulation 1** | **Output Simulation 1** |
|---|---|
| Initial diameter $d_0$ and initial height $h_0$ of the specimen, ambient temperature $T_a$, furnace temperature $T_F$, time-temperature amplitude for the temperature in the furnace, heating time $t_h$, emissivity of furnace and specimen material $\varepsilon_s$, heat transfer coefficient for convection $h_c$, thermal contact conductance $k_c$, global seed size for specimen and furnace, file path for the output database, name of the simulation/job | .odb file<br>Field output: NT, HFL<br>History output for defined node set: NT, COORD |
| **Input Simulation 2** | **Output Simulation 2** |
| Initial diameter $d_0$ and initial height $h_0$ of the specimen, ambient temperature $T_a$, temperature of the specimen at the end of the previous heating simulation $T1_{end}$, transport time $t_t$, emissivity of specimen material $\varepsilon_s$, heat transfer coefficient for convection $h_c$ | Temperature at the end of the transport $T2_{end}$ |
| **Input Simulation 3** | **Output Simulation 3** |
| Initial diameter $d_0$ and initial height $h_0$ of the specimen, process time, emissivity of the specimen material $\varepsilon_s$, heat transfer coefficient for convection $h_c$, ambient temperature $T_a$, temperature of the specimen at the end of the previous transport simulation $T2_{end}$, thermal contact conductance $k_c$, global seed size for specimen and bottom die, file path for the output database, name of the simulation/job | .odb file<br>Field output: NT, HFL<br>History output for defined node set: NT, COORD |
| **Input Simulation 4** | **Output Simulation 4** |
| Initial diameter $d_0$ and initial height $h_0$ of the specimen, time for the upsetting process, friction coefficient μ, thermal contact conductance $k_c$, file path to output database of previous simulation, time-displacement amplitude, global seed size for specimen and dies, file path for the output database, name of the simulation/job | .odb file<br>Field output: S, U, PE, PEEQ, CSTRESS, CFORCE, NT, HFL<br>(+ DAMAGEC, DMCRT, for explicit simulation)<br>History output for defined node set: NT, COORD<br>History output for reference point: U3, RF3 |

All simulation and process parameter are specified at the beginning of the ‘main_script.py’ and are passed on to the respective function to run the simulation. An overview on the input and output parameters for each simulation is given in Table 20. Additionally, each module provides comments with information on the necessary input variables. Specified field or history output variables are defined in the Abaqus Documentation [31].

# 8. Evaluation and Results

In this chapter, sensor data provided by the CPPSs during the experiments is discussed. Thereby, the focus is not only on the measurements within a test setting, but rather on the comparison of the data between different settings. Additionally, the condition of the specimens after forming is discussed. Further, input values and influencing factors on the simulations are outlined and simulations are evaluated. Prior to presenting the differences between experiment and simulation, challenges, occurred with automated simulation models are mentioned. Finally, an overview is given on the comparison between the experiments and the results generated with the automated simulations.

## 8.1. Interpretation of the sensor data

An example for the visualization of the measured quantities is given in Figure 33. By using the ‘measurement.py’ script, this plot is created for each test setting including the corresponding measurements. The visualization serves identify significant divergences to further exclude outliers. Additionally, it can be used for plausibility checks regarding the sensor data. In the following measurements belonging to different test settings are compared with each other, to analyze influencing factors on the process cycle. Thereby, influence of temperature, transport time, upset height and preheating temperature is assessed.

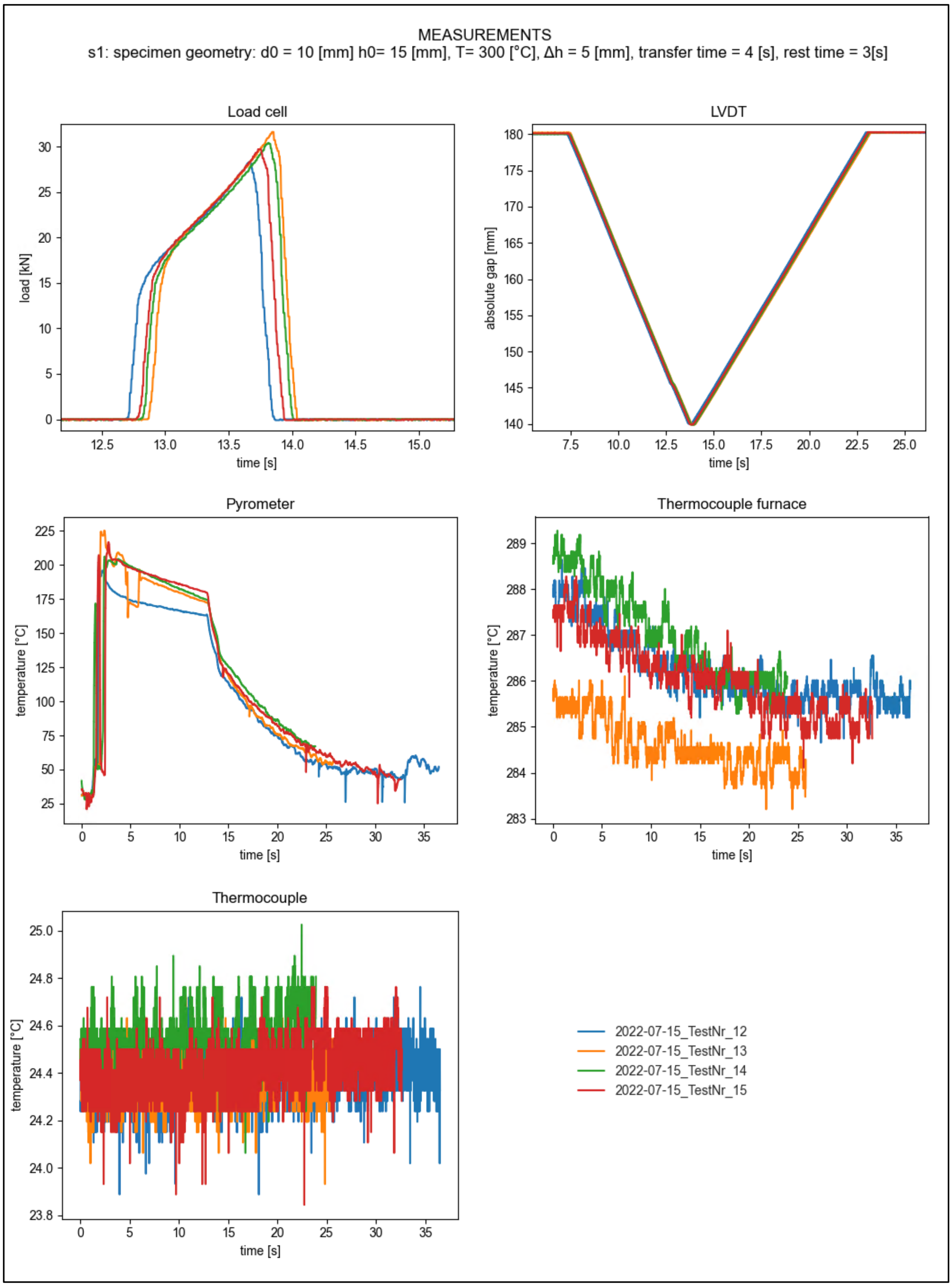


**Figure 33**: Visualization of sensor data

To demonstrate the strong dependence of the upsetting force on the specimen temperature, test settings with the same geometry, transport time, and upset height were compared. Temperature curves are shown in Figure 34, and corresponding upsetting forces are illustrated in Figure 35. Thereby, the predefined furnace temperatures are 300 °C (blue), 400 °C (orange) or 500 °C (red). Lower specimen temperatures correlate with higher upsetting forces.

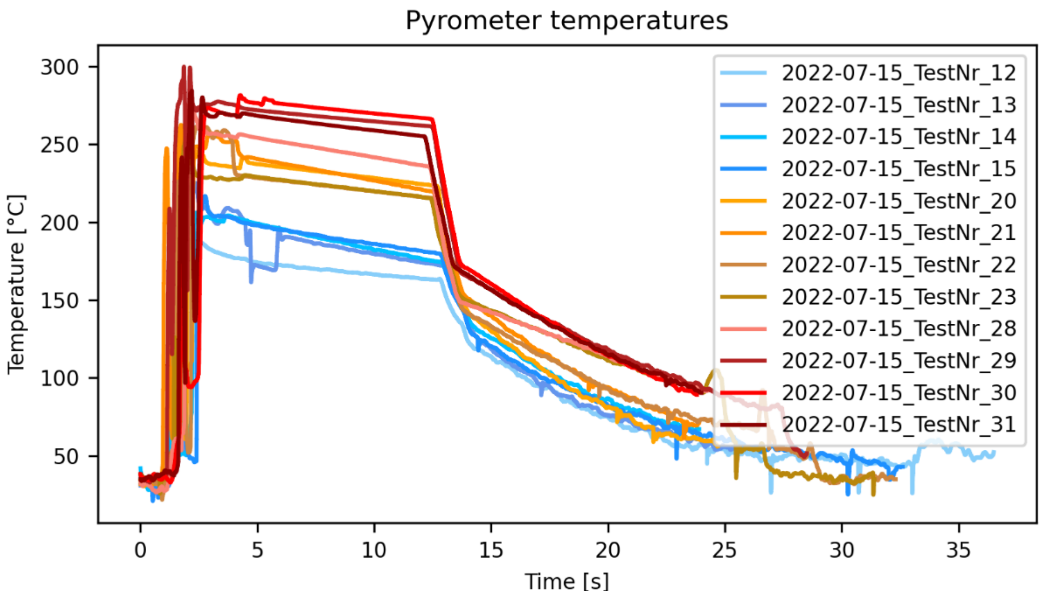


**Figure 34:** Temperatures (experiment 2, geometry A, $t_t$ = 4 s, Δh = 5 mm)

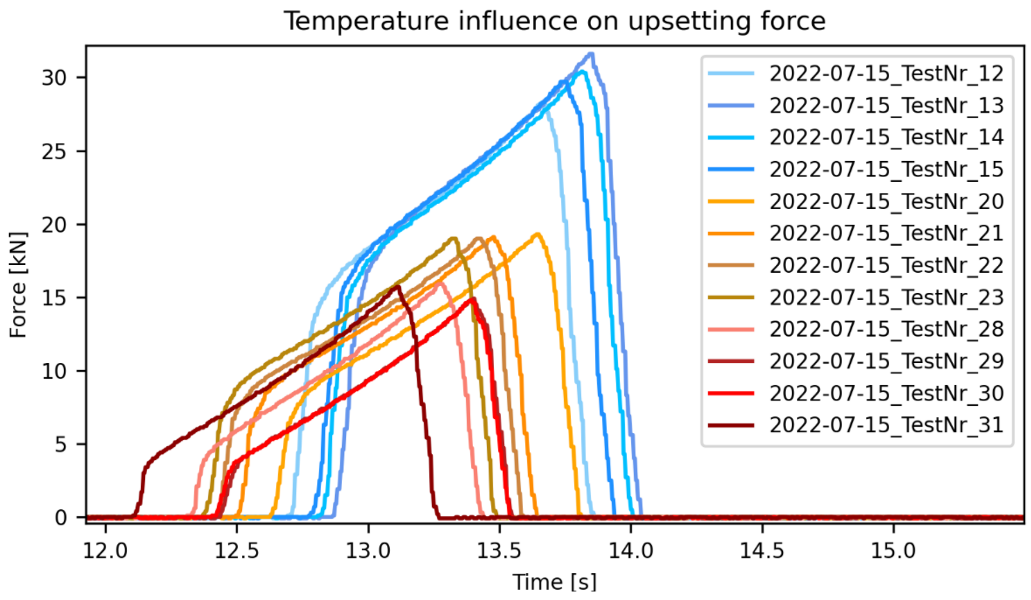


**Figure 35:** Upsetting force (experiment 2, geometry A, $t_t$ = 4 s, Δh = 5 mm)

Measured specimen temperatures between 160 °C and 180 °C at the time before the upsetting process starts result in an average force of 30 kN. Force is reduced by the factor two as the pyrometer detects specimen temperatures between 235 °C and 270 °C, which shows a high dependence of the force on the specimen temperature.

**Figure 36** illustrates the force-time curve for test setting s5 with an upset height of 5 mm, shown in green and test setting s6 with an upset height of 8 mm, shown in blue. Furnace temperature is set at 500 °C for both settings, whereas transport time varies. Force goes up as the upset height increases.

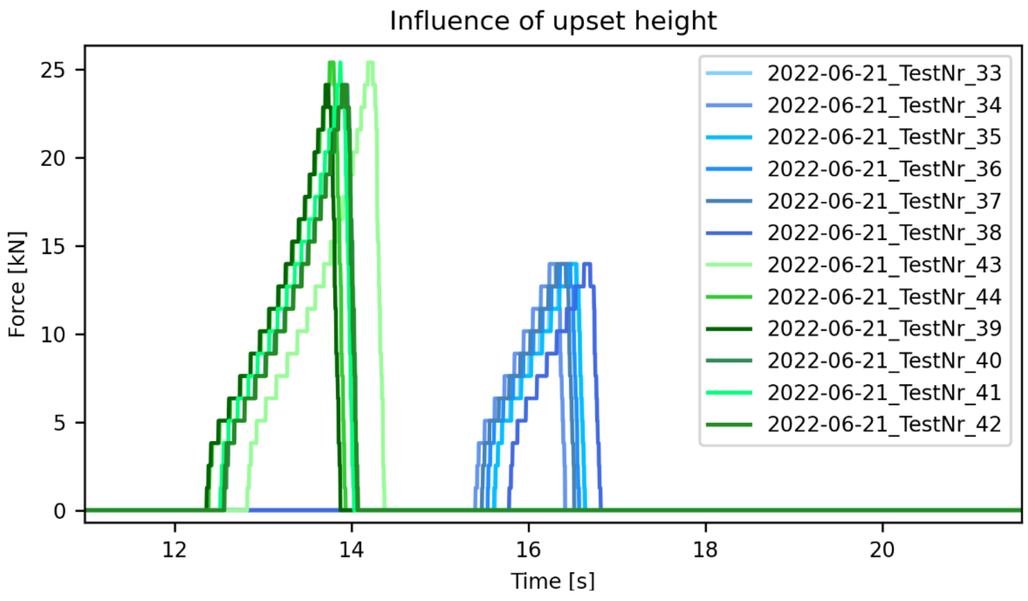


**Figure 36:** Upsetting force (experiment 1, geometry A, $T_F$ = 500 °C, $t_t$ = 4 s / 7 s, Δh = 5 mm / 8 mm)

Further, both settings show a linear temperature curve, see Figure 37, during the contact to the bottom die prior to the compression of the specimen. Temperatures at the point of time the upsetting starts are not significantly lower for a transport time of seven seconds. Therefore, the influence of the transport time on the upsetting force is low. After the upsetting, an increase in the specimen temperature is visible for the specimen compressed to more than half of the initial height.

Within the entire process, time restrictions regarding transport time and rest time were met very well. The LVDT measurements show time differences of less than a second between measurements of the same test setting before the die moves downwards. Equally, this can be observed in the measurements of the load sensor.

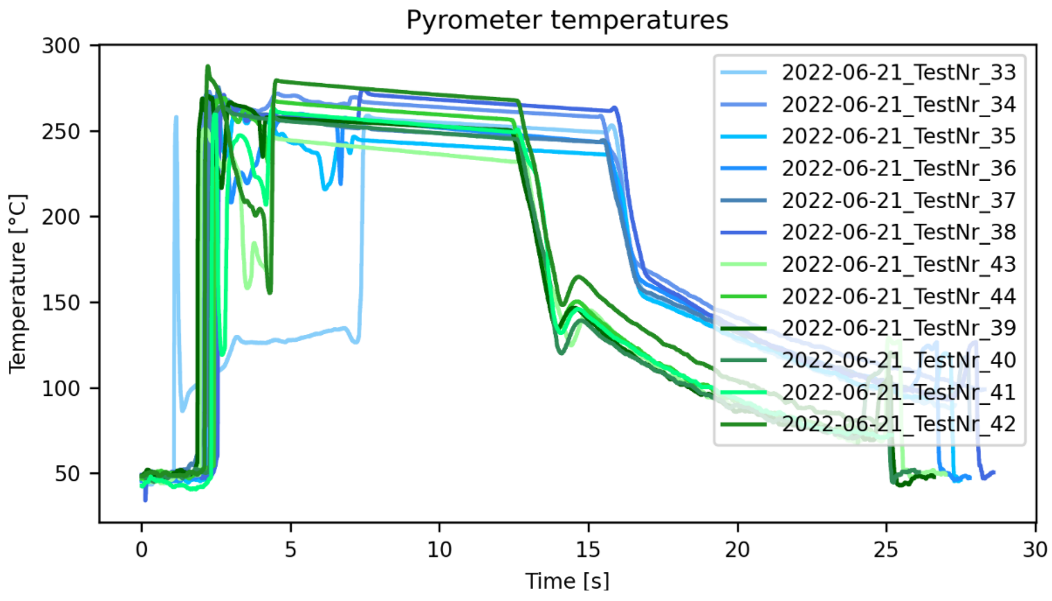


**Figure 37:** Pyrometer temperature (experiment 1, geometry A, $T_F$ = 500 °C, $t_t$ = 4 s / 7 s, Δh = 5 mm / 8 mm)

Subsequently, experiment 1 and 2 are compared to each other, on the example of test setting s1, which was the same in both experiments. As expected, upsetting force, depicted in Figure 38, is approximately the same for setting s1 in experiment 1, illustrated in blue and experiment 2, shown in orange.

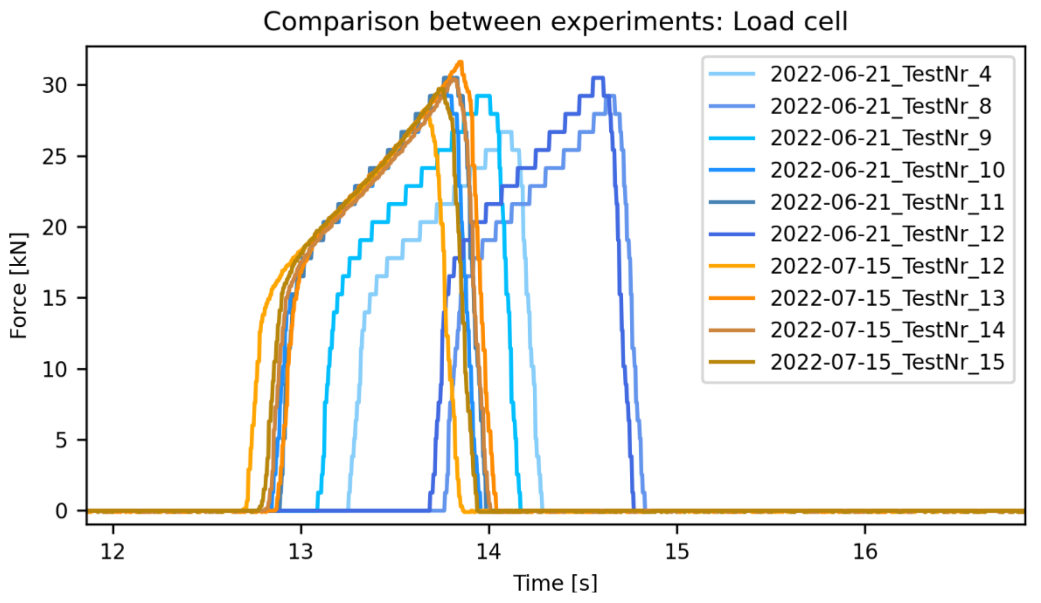


**Figure 38:** Comparison between experiments with the same test setting (s1): Load cell

A difference between the two experiments was the positioning of the pyrometer. In the first experiment, the pyrometer position was on the left side of the hydraulic press. Temperature measurements from the pyrometer are illustrated in Figure 39. Fluctuations in the first view seconds of the measurement occur, due to movement of the gripper, which occasionally crosses the measuring position in experiment 1. During the second experiment, the pyrometer position was on the back side of the hydraulic press. Changes in the position of the pyrometer lead to lower fluctuations around the time the specimen is placed on the bottom die. However, this does not influence the temperature measurement during upsetting.

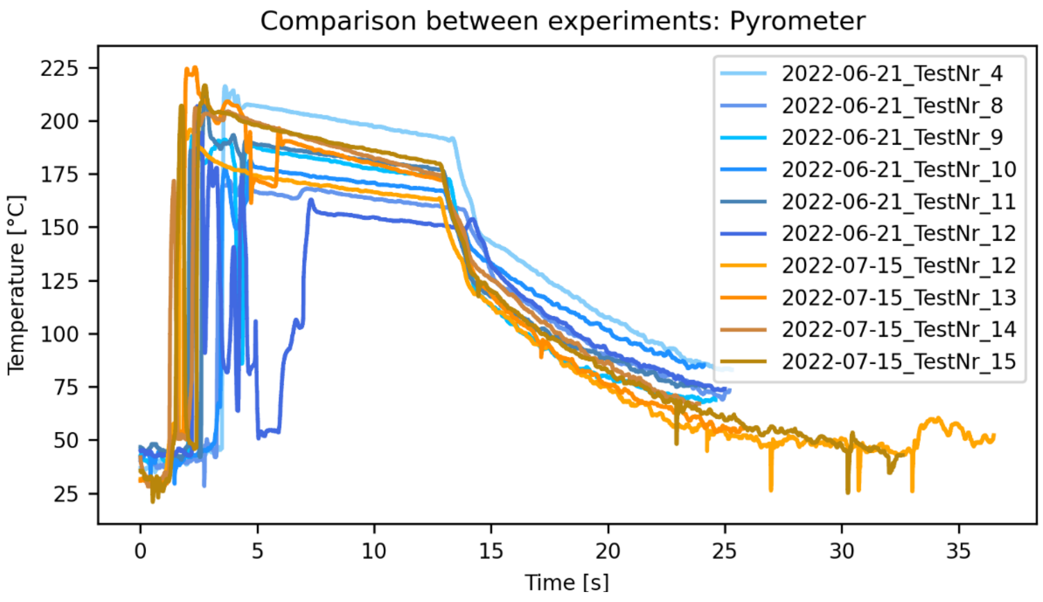


**Figure 39:** Comparison between experiments with the same test setting (s1): Pyrometer

Even though, the predefined furnace temperature is the same for both settings, pyrometer measurements show a wider temperature range for the test setting s1 in the first experiment. This could be due to the furnace temperatures, which show a wider temperature range than in the second experiment, see Figure 40. Furthermore, discrepancies between the temperature measurements of the preinstalled thermocouples and the retrofitted thermocouple occurred. As the preinstalled thermocouple, which is connected to the internal control system of the furnace, measured the predefined temperature, the furnace stops heating up. However, the retrofitted thermocouple, which is connected to the HMI, measured lower temperatures. Reference temperatures were taken from the retrofitted thermocouple, as no other data is available. According to this sensor, the predefined temperature was not reached during the experiments. These observations were made for all furnace temperature measurements.

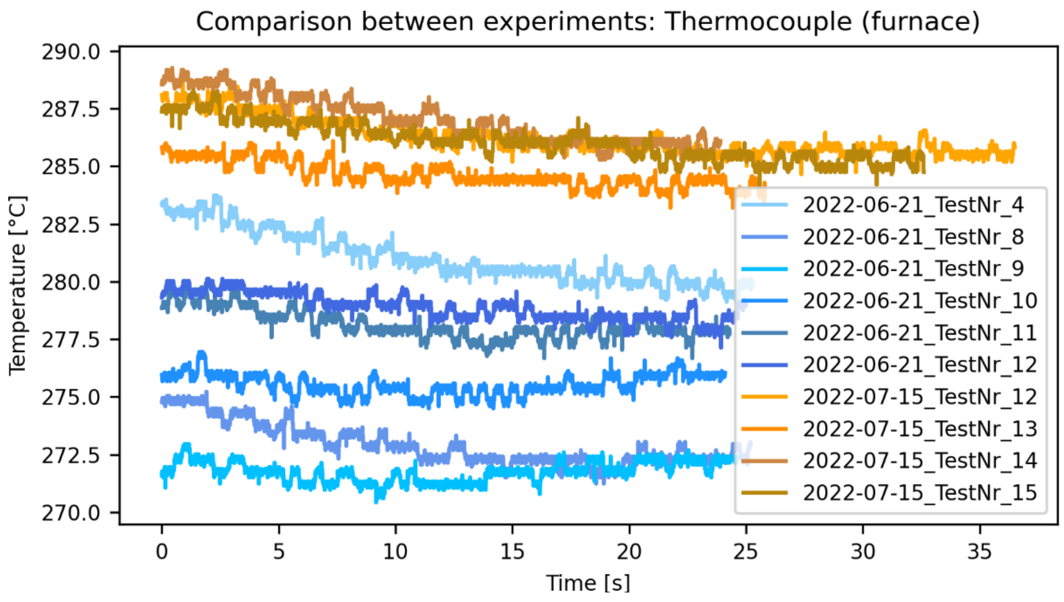


**Figure 40:** Comparison between experiments with the same test setting (s1): Thermocouple (furnace)

In metal forming it is important, whether the desired end geometry of the specimen can be accomplished without failure of the material. Figure 41 shows four of the specimens tested in experiment 1. The specimens show dissimilar surface texture, depending on the specimen geometry. The surface of the smaller specimens A, illustrated on the left side, is rough, and cracks occurred 45° to upsetting direction. In contrary, the bigger specimens B, shown on the right side, have an even surface and cracks 0° to upsetting direction are detected. Further information on the specimen geometry before and after forming and whether visible cracks occurred, is given in Appendix L for both experiments.

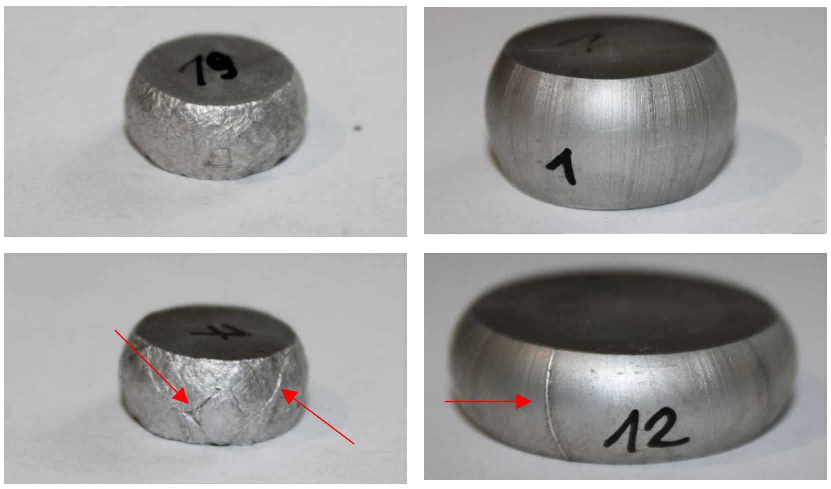


**Figure 41:** Specimen after forming

## 8.2. Influences on the simulations

In the following influencing parameters on the four simulations are evaluated and discussed. Furthermore, the calibration of input parameters is outlined, and simulation results are presented.

### 8.2.1. Heating

During the heating of the specimen, measurements show temperature drops as the furnace is opened and heat exchanges with the environment. For reasons of simplicity, the heating simulation considers a constant temperature during the entire heating time. This temperature is defined by the average furnace temperature calculated from the thermocouple measurements for each test setting, listed in Table 21.

**Table 21:** Furnace temperatures

| | Setting name | Predefined temperature [°C] | Average temperature during a test setting [°C] |
|---|---|---|---|
| Experiment 1 | s1 | 300 | 276 |
| | s2 | 300 | 279 |
| | s3 | 300 | 281 |
| | s4 | 300 | 279 |
| | s5 | 500 | 473 |
| | s6 | 500 | 476 |
| | s7 | 500 | 479 |
| | s8 | 500 | 480 |
| Experiment 2 | s1 | 300 | 286 |
| | s2 | 300 | 285 |
| | s3 | 400 | 376 |
| | s4 | 400 | 377 |
| | s5 | 500 | 475 |
| | s6 | 500 | 476 |

Figure 42 depicts the heating curve of a node on the outer surface of the specimen predicted by the heating simulation. For longer heating periods, the specimen temperature gets closer to the predefined furnace temperature. The specimen temperature after 30 minutes of heating correlates well with the average temperature measured in the furnace during a test setting. The curve depends on the specified values for emissivity, heat transfer coefficient for convection and thermal contact conductance, which were estimated. The emissivity of the refractory material is defined as 0.8, the emissivity for the specimen is defined as 0.3. Heat transfer coefficient for convection is estimated from the reverence values for free convection in Table 19. Thermal contact conductance is defined as in simulation 3. By using lower values for these three parameters, it takes the specimen longer to heat up.

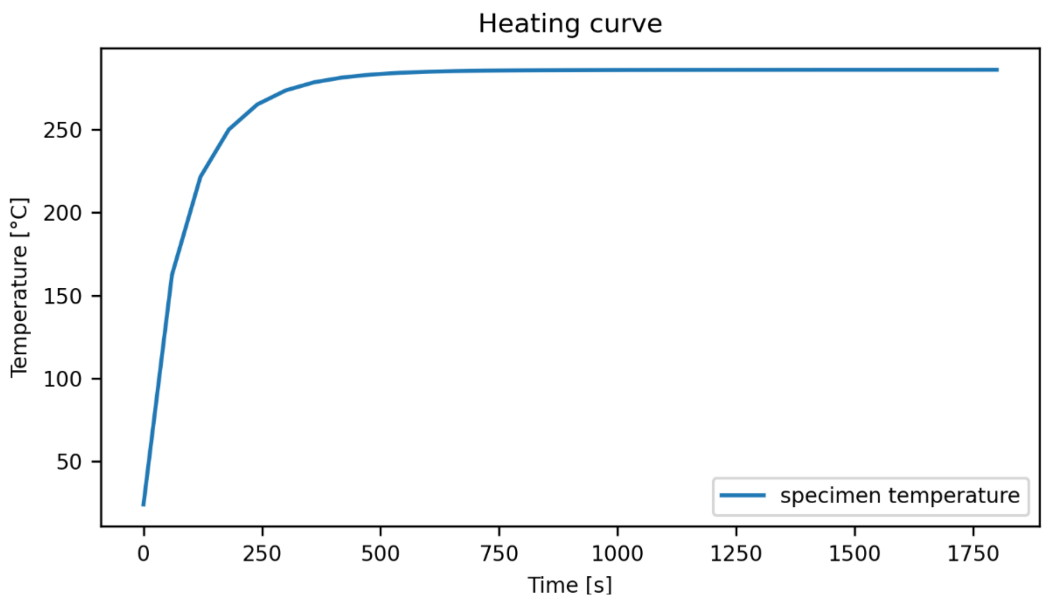


**Figure 42:** Heating curve - simulation 1

Measurements were made to approximately determine the specimen temperature after a defined heating time. Therefore, the pyrometer was positioned in front of the furnace and a specimen was removed from the furnace after a defined heating time. Overall, eight specimens were used, whereas each specimen remained in the heating chamber of the furnace for four more minutes than the previous one. Average furnace temperature was at 280 °C during the test. To reduce heat loss, the specimens were directly placed on a steel plate in front of the furnace. After four minutes the pyrometer measured a peak temperature of around 200 °C. A maximum temperature is detected after a heating time of around 25 minutes.  The temperature curves in Figure 43 show, that the peak temperature measured for each specimen does not increase from specimen to specimen with increasing time. This might imply, that the positioning of the specimen in the furnace has an impact. On the other hand, this temperature differences can be related to differences in timing due to rapid cooling when taking the specimen out of the furnace. Further experiments, to adapt the heating curve to the experiment were not made. To predict the heating time more precisely, further experiments with varying heating time are necessary, to determine temperature distribution during heating.

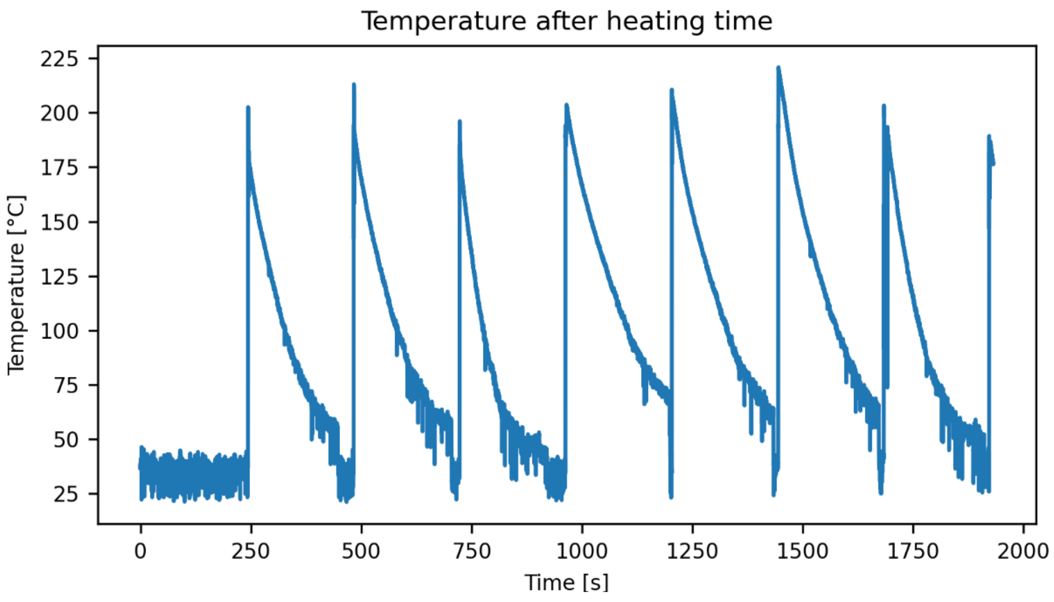


**Figure 43**: Specimen temperature after removing from the furnace

### 8.2.2. Transport

There is no temperature measurement available until the specimen is placed on the bottom die of the hydraulic press. For this reason, only assumptions can be made to determine temperature distribution during transport. Temperature curve is estimated through reference points obtained from the measurements. Temperature at the beginning of the transport simulation is assumed to be equal to the average furnace temperature of measurements from the same test setting. Temperature at the end of the transport simulation is assumed to be equal to the temperature measured from the pyrometer after four or seven seconds of transport, depending on the experimental plan. However, fluctuations occur in the measurements during this time if, for example, the gripper crosses the measuring position. Using this temperature values after the transport time as a reference could lead to uncertainties. Therefore, temperatures at the time the specimen first contacts the bottom die are calculated by using a linear fit.

Referring to Figure 22, measurements show a linear heat loss during the contact time to the bottom die between timepoint $t_1$ and $t_2$. Measurement values starting after half of the contact time until the start of the upsetting are used as input values for a linear polynomial fit, illustrated in Figure 44. Temperature measurements in the first half were excluded as this would lead to deviations in the gradient of the curve. Using the linear fit, the temperature at the beginning of process step 3 is determined for further use as a reference value in the transport simulation. Thereby, this temperature value refers to the specimen temperature after four or seven seconds of transport, depending on the experimental plan.

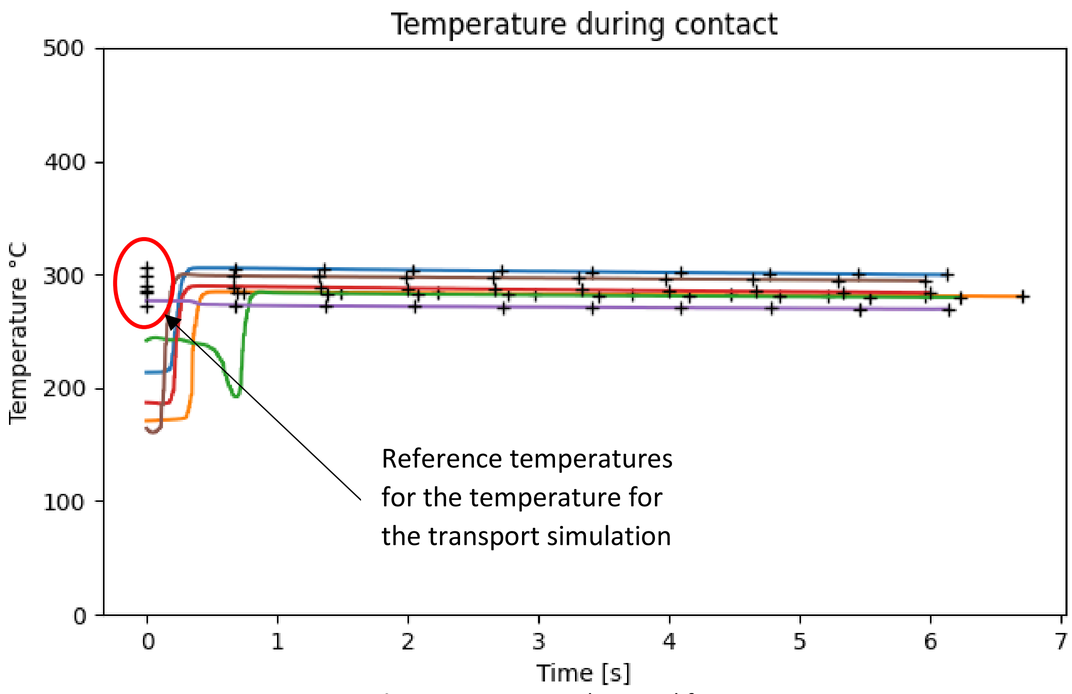


**Figure 44:** Linear polynomial fit

Reference points obtained from the experiments are used to fit the temperature curve determined in simulation 2, which is illustrated in Figure 45. Thereby, a steep drop between the specimen temperature at the end of the heating process until the specimen is placed on the bottom die after four seconds of transport occurs. On the contrary, the decrease in temperature between four or seven seconds of transport is quite low. The discretized energy balance equation (5.28) is used to determine the temperature curve, considering heat flux, convection, and radiation.

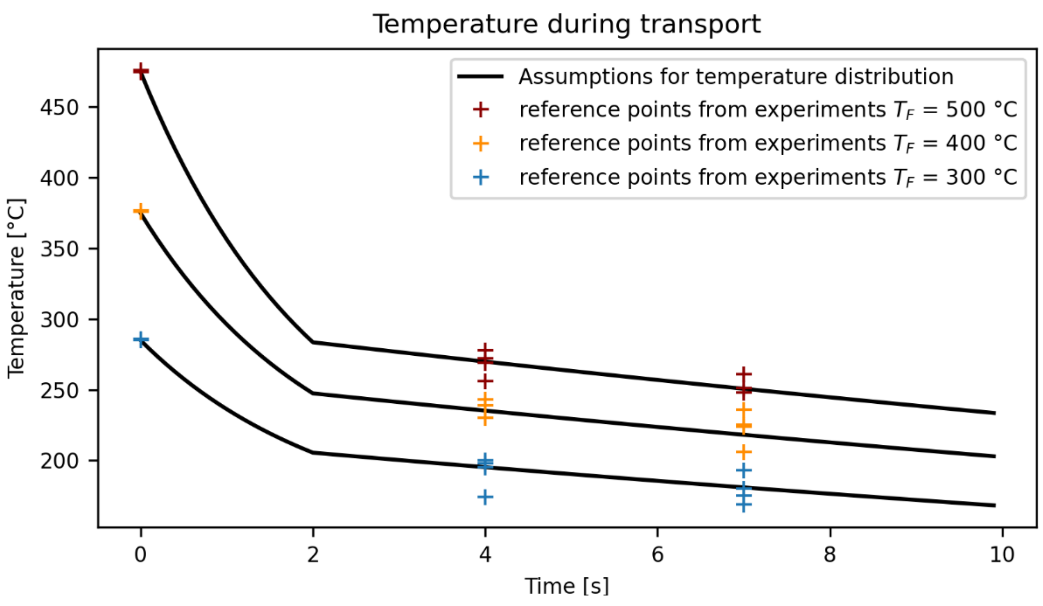


**Figure 45:** Reference values for the transport obtained from experiment 2

For simplicity, a correction term, depending on the initial temperature and the specimen geometry, is introduced for the heat flux term in the first two seconds of the transport to fit the curve to the measurements. The rest of the time, only convection and radiation are considered. As heat loss due to radiation is rather small, the heat transfer coefficient for convection can be used to adapt the slope of the curve. The assumption was made, that heat loss is significant during the first seconds as the specimen is removed from the furnace. Additionally, temperature loss decreases as the difference between specimen and ambient temperature drops. Within this approach are some errors from a physical point of view. The minimum transport time in practical is approximately three seconds to move the specimen to the press, therefore the previous temperature curve is not very important.

### 8.2.3. Rest on die

During the contact to the bottom die, heat transfers to a large amount to the die, while heat loss due to convection and radiation is negligible. Therefore, different values for the thermal contact conductance were tested to adjust the temperature curve to the slope shown in the measurements. Reference values, mentioned in Table 18, are too high, as the contact pressure during this process step is low. To compare the impact of different values, the temperature profile is evaluated for the node on the shell surface six millimeters above the bottom die surface, which is equal to the measuring position of the pyrometer, see Figure 46. After constantly decreasing the values, good correlations are found with a thermal contact conductance of 0.3 mW/mm$^2$°C, which is used in further simulations.

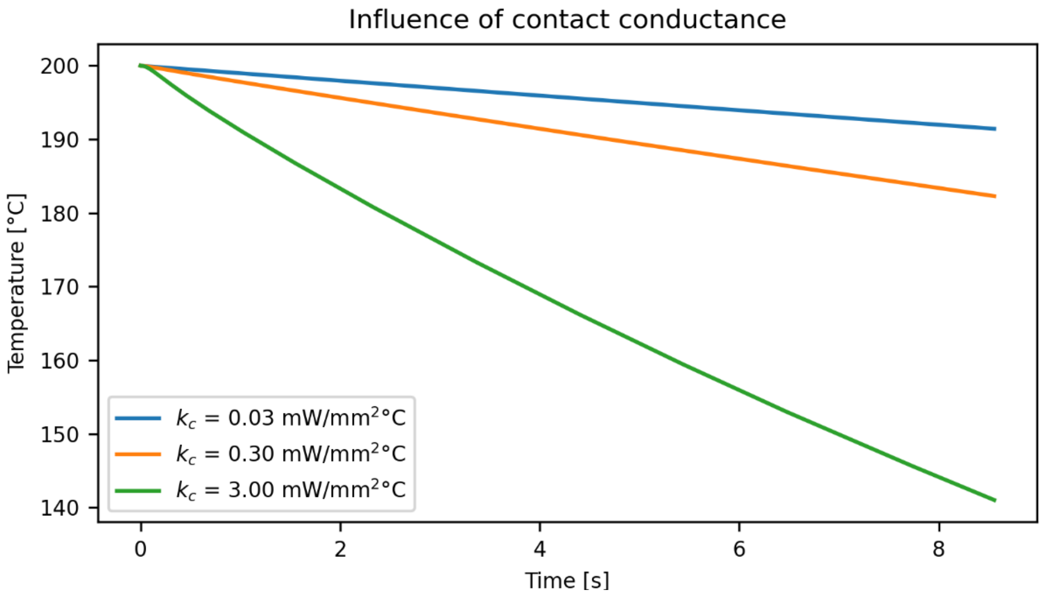


**Figure 46**: Influence of contact conductance

FE simulations reveal that the temperature distribution in the specimen is quite uniformly during the entire process time. Figure 47 depicts the temperature distribution in the cut surface of a specimen of geometry A and an initial temperature of 200 °C after a contact time of 8.5 seconds to the bottom die. Temperature differences less than 4 °C are present, while temperatures at the surface area are slightly higher than within the center of the specimen. Comparable results were found for specimen of geometry B. This shows that convection and radiation to the environment have a negligible impact. However, computation time is about less than a minute, and no significant rise occurs if conduction and radiation are considered.

Summarizing, temperature drop during this simulation can be easily adapted to the measured pyrometer temperatures by varying the thermal contact conductance between specimen and die. Anyway, the temperature determined at the end of the transport simulation affects the temperature distribution as it shifts the curve to higher or lower values.

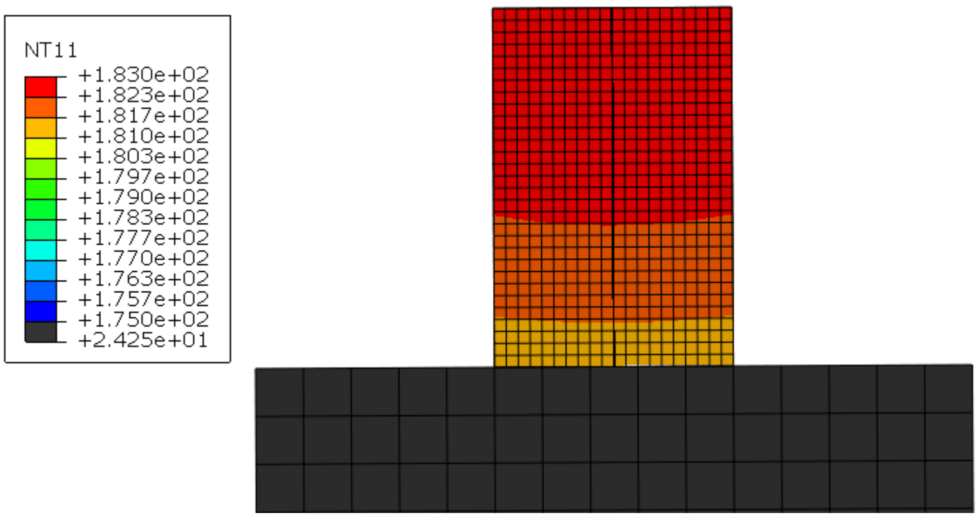


**Figure 47:** Temperature distribution after a contact time of 8.5 s

### 8.2.4. Upsetting

Different influencing factors on the upsetting simulation have been investigated and the force-displacement curve during upsetting was evaluated. The following simulations were carried out for a specimen geometry A, a constant initial specimen temperature of 175 °C, an initial temperature of 24 °C for the dies and Johnson-Cook parameters from [45], listed in Table 16.

First, implicit and explicit simulation are compared with each other. Figure 48 shows a good agreement between implicit simulation with a maximum force of 38.55 kN and explicit simulation, predicting a maximum force of 38.76 kN. Due to the small stable time increment computation time for the explicit simulation is higher than for the implicit simulation. To reduce computation time, mass scaling can be applied to the explicit simulation. For further evaluations in this work, the implicit simulation was used.

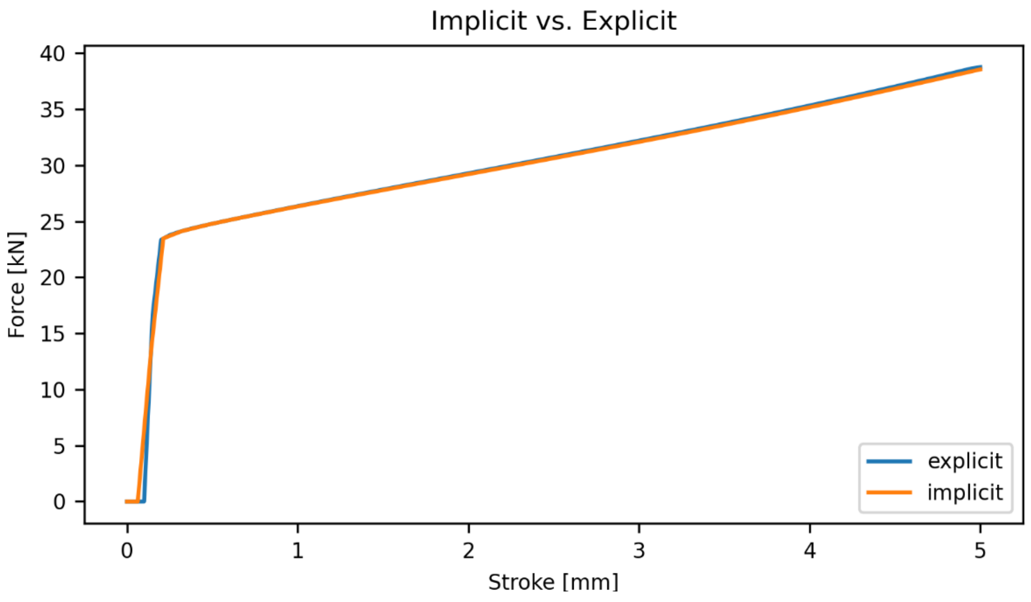


**Figure 48:** Implicit vs. Explicit

Influence of mesh size was evaluated considering three different combinations of element sizes, defined by global seed size. Accuracy increases with decreasing mesh size, simultaneously, computation time rises. However, too small elements result in high computation time, without relevant improvement in accuracy. As illustrated in Figure 49, for the tested geometry, a mesh size of

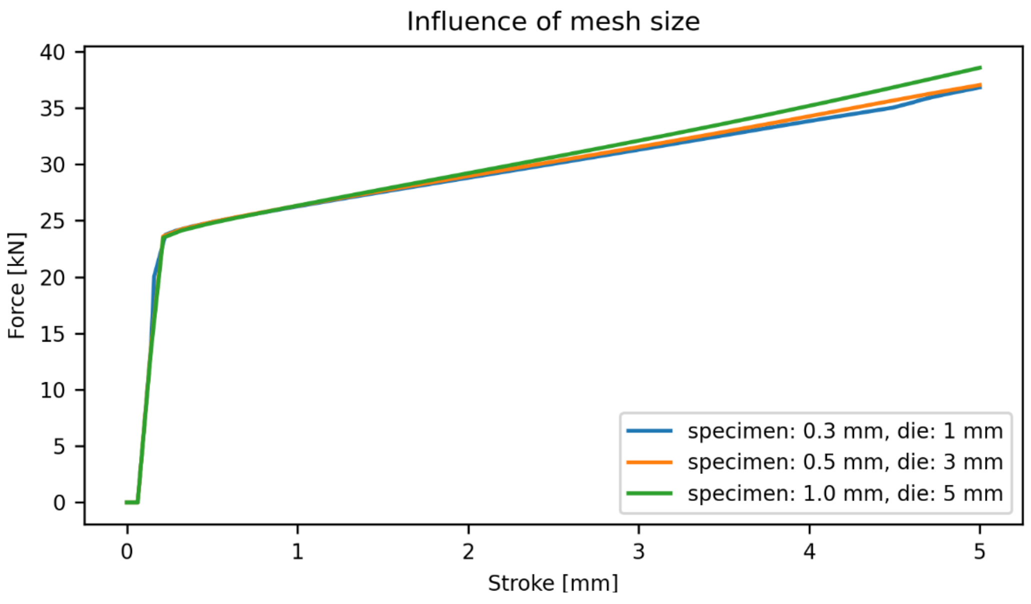


**Figure 49:** Influence of mesh size

0.5 mm for the specimen is sufficient. As the model is automated to run simulations for different specimen geometries, the influence of mesh size needs to be evaluated for further geometries.

Furthermore, different parameters for the Johnson-Cook material model were studied as the parameters found in the literature vary. The force-displacement curve highly depends on the Johnson-Cook parameters. Figure 50 depicts the impact of parameters 1-5 obtained from literature, listed in Table 16. Thereby, the maximum force deviates from 43 kN to 34 kN for an initial temperature of 175 °C. The decrease in force for a starting temperature of 250 °C is not significant.

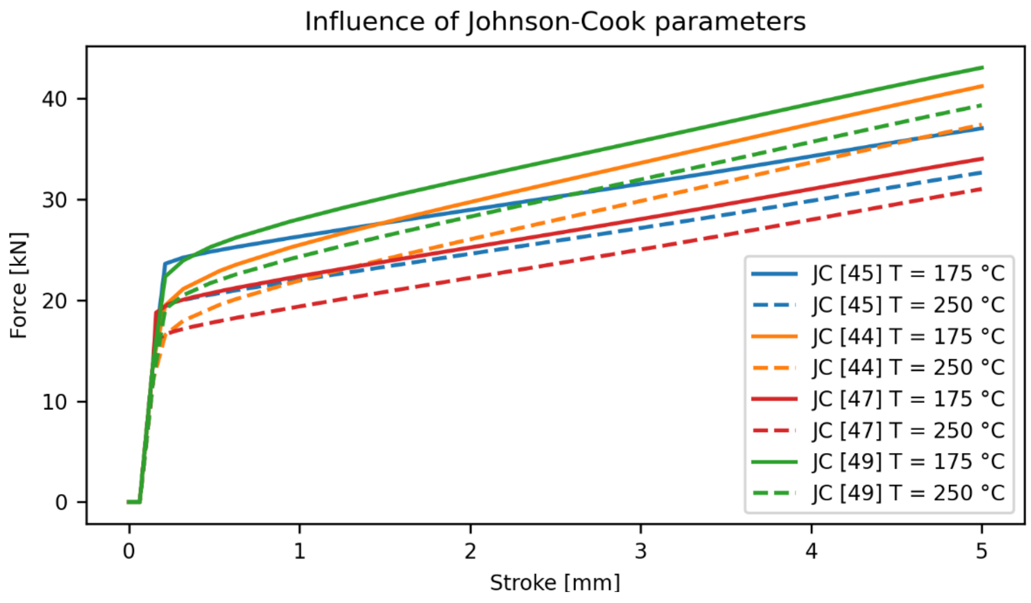


**Figure 50**: Influence of different Johnson-Cook parameters

For the automated simulations, Johnson-Cook parameters were used from [47] as other model parameter tend to predict higher forces, thereby the thermal softening coefficient was adapted to $m = 0.9$. Figure 51 illustrates, stresses, strains, and temperatures for the upsetting simulation of setting s3 after a compression of 15 mm.

In the contact area between specimen and die, low deformation occurs due to the friction to the dies, high plastic deformations occur in the middle of the specimen. Friction coefficient is assumed as $\mu = 0.3$. Temperature of the specimen is lower on the contact surfaces to the dies. Temperatures differences between the center and the outer surface of the specimen are low, as the specimen dimensions are small, and the material has a high conductivity. Reference values for the thermal contact conductance, listed in Table 18, were used and further adapted, to fit the temperature at the measuring position to the measurements. A value of 20 mW/mm$^2$K was used, this value can be further adapted, to fit the temperatures to the pyrometer measurements. Thereby, higher values for the thermal contact conductance lead to higher heat transfer to the die, leading to lower specimen temperatures.

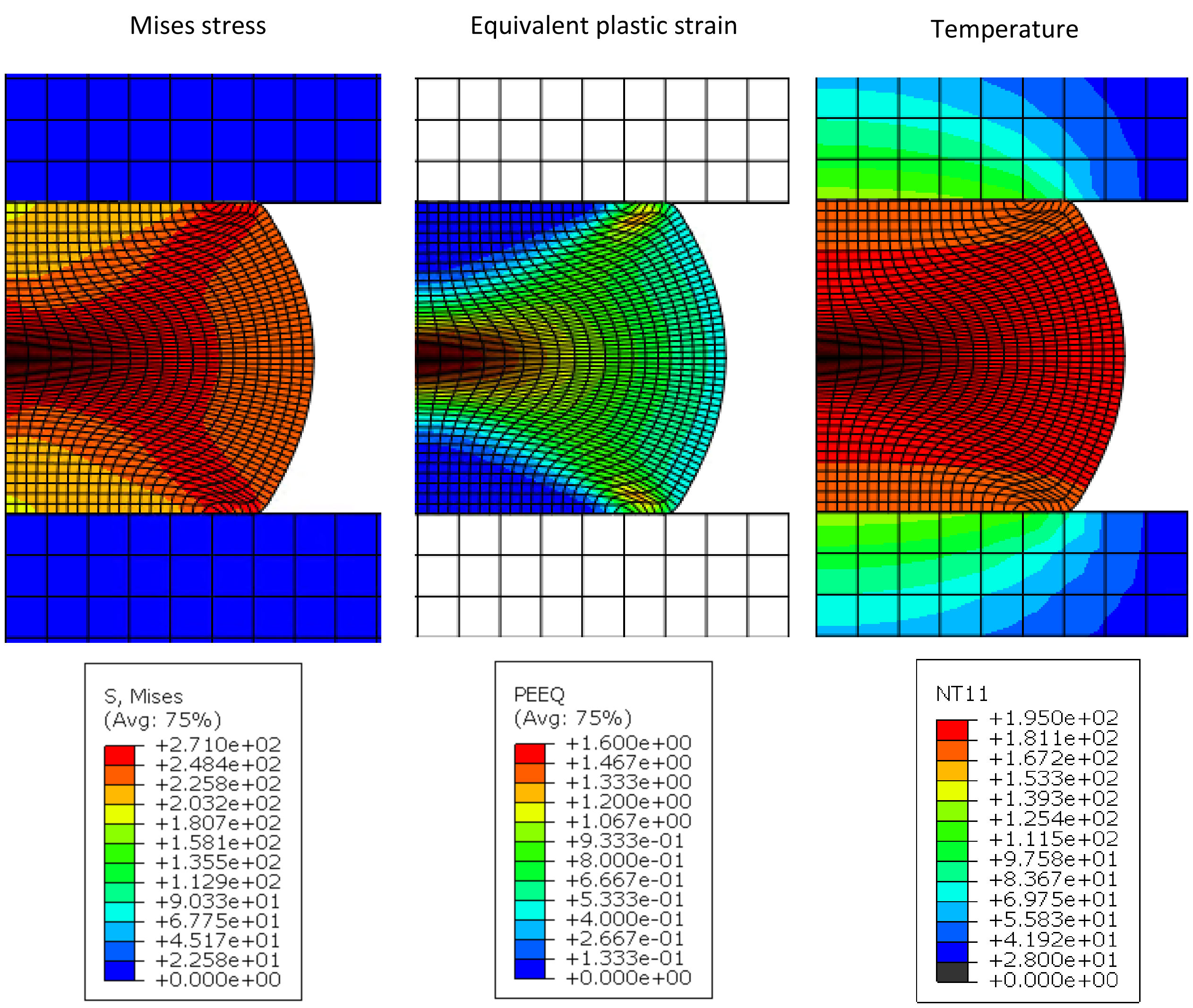


**Figure 51:** Upsetting simulation for setting s3

## 8.3. Challenges with automated models

At the beginning it is crucial to specify the purpose and the application of the models under the consideration of potential changes, which might be arising during the development. Otherwise, this could lead to problems during adapting or expanding the script. To implement changes occurring during the development process, a modular script structure is beneficial and provides the possibility to replace or modify modules without significant effort. Furthermore, variable parameters need to be restricted to necessary process parameter, as a vast number of input parameter is quite confusing. For instance, the naming of the parts, instances, boundary or initial conditions is hard coded, as it is not relevant, whereas heat transfer coefficient or thermal contact conductance are unclear process parameter, which need to be adaptable. Also, using flexible functions can be beneficial if the functions fit the purpose and are not too complex. An example is a function that was created to select a specific surface, which is identified by the coordinates of a point on this surface.

Sanity checks are important to verify the entered parameter, to eliminate errors caused by users. So far, the main script does not include sanity check, thus it would be beneficial to add them, as by using automated simulations, errors might not be quite obvious and easy to discover, because error messages do not necessarily describe the source of the error. For instance, if the user accidentally enters a value higher than the initial height of the specimen for the height after forming in the main script, an error occurs. The script calculates the height difference and uses the entered velocity and the height difference to calculate the step time, which is negative in this case. To create a step, the step time needs to be greater than zero. An error message occurs as the step generation failed, but no further information is given. However, the origin of the error might be difficult to discover.

Moreover, too large mesh size leads to inaccurate results, or specific process parameter need to be within a certain limit. To avoid errors, a documentation including reference parameters is beneficial. The flexibility of an automated simulation is a benefit on the one hand but can be a major drawback on the other hand if used in a wrong way. In fact, the specification of limits is necessary to guarantee the right use. As an example, the transport simulation assumes, that the temperature change of the specimen is homogeneously in the whole volume. If the specimen is too large and the material has a low conductivity, this approximation is not valid anymore, which is why limits are needed to avoid wrong results. A complete documentation of the scripted models is necessary for traceability.
Within this work a basic structure was developed to automate simulations. However, further validation on material model and simulation parameter and assessment of influencing factors are indispensable.

## 8.4. Comparison between experiment and simulation

By using the Python 'compare.py' script, the following evaluation, depicted in Figure 52, is created for each test setting. This serves as a basis to automatically compare simulation and experiment. The Figure shows, that for the test setting s1 the temperature curves as well as the force-displacement curves fit well. Due to the thermal expansion the displacement during the stroke is higher than in the simulation, which does not account thermal expansion. Therefore, also the process time during upsetting is higher in the experiment.

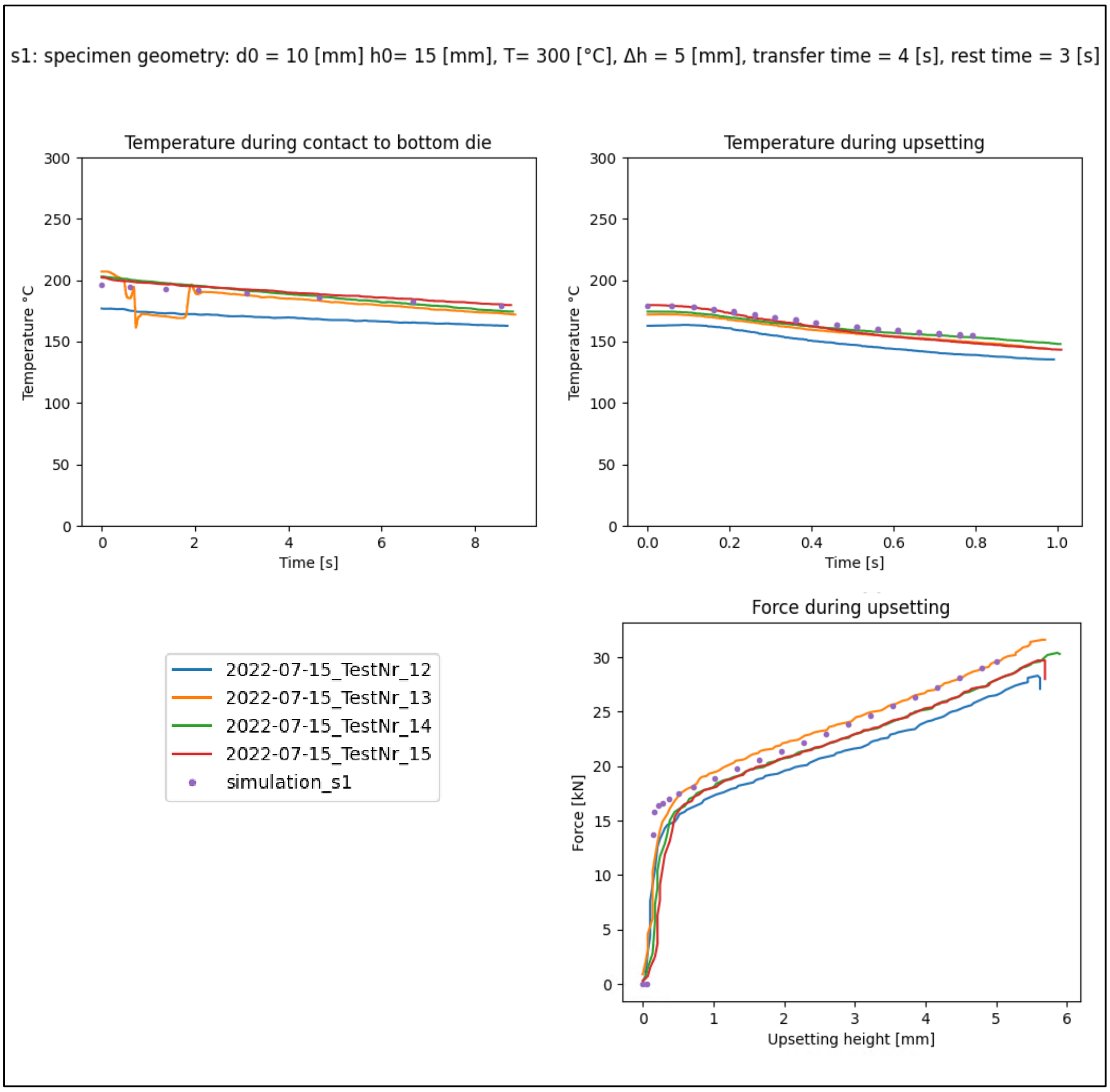


**Figure 52:** Comparison between experiment and simulation

In Table 22 and Table 23 the results of the automated simulation sequence are summarized. Therefore, the average maximum force during tests is compared to the maximum force predicted in the simulation. Further, it is assessed, how the temperature curve detected in the simulation fits the pyrometer measurements from the experiments. Overall, temperature distribution from simulation 3 and simulation 4 show good agreement with measured temperatures. However, large differences in upsetting force occur, especially for higher specimen temperatures. The high dependence of the specimen material, demonstrated on the measurements, is not predicted in the simulations. Experiments show a force reduction of approximately the half at higher temperatures, whereas in the simulation with the force does not drop significantly.

**Table 22:** Comparison between experiment 1 and simulation

| Setting name | $F_{max}$ [kN] Experiments | $F_{max}$ [kN] Simulation | Deviation [%] | $T(t)$ during Simulation 3 | $T(t)$ during Simulation 4 |
|---|---|---|---|---|---|
| s1 | 29.42 | 29.19 | - 1 | * | * |
| s2 | 39.88 | 45.29 | + 13 | * | * |
| s3 | 133.98 | 151.25 | + 12 | * | * |
| s4 | 218.86 | 287.29 | + 31 | * | ** |
| s5 | 13.97 | 26.17 | + 87 | * | * |
| s6 | 24.77 | 41.27 | + 66 | * | * |
| s7 | 62.02 | 137.28 | + 121 | * | ** |
| s8 | 127.85 | 262.42 | + 105 | * | * |

* Good agreement, within temperature measurements

** Lower temperatures (max. 20 °C)

It is assumed, that the Johnson-Cook material model has a significant influence on the force-displacement curve. Also, parameters in the literature differ, depending on the test method and the test temperature range. For further improvement of the model, the Johnson-Cook parameters need to be determined from experiments, considering the same material, similar specimen geometry, compressional loads, and the same temperature range.

Additionally, the core temperature of the specimen could be significantly higher than the temperatures measured on the surface area of the specimen. Consequently, higher core temperatures of the specimen lead to lower forces during the upsetting process, as the flow stress decreases with higher temperatures. This means that the simulation models for the transport (simulation 2) and the rest time (simulation 3) do not accurately describe the temperature distribution inside the specimen and need to be adapted.

Further, the specimen material could deviate from specified material properties. Therefore, test with specimen of the same geometry but another batch of material can be used to compare the results. For some of the specimen, cracks occurred during the forming process. Material failure could also decrease the upsetting force, thus is not accounted in the simulation. Additionally, tests can be conducted to examine the used specimen if cracks occurred inside the material.
Also, the pyrometer can be checked to ensure validity of the data.

**Table 23:** Comparison between experiment 2 and simulation

| Setting name | $F_{max}$ [kN] Experiments | $F_{max}$ [kN] Simulation | Deviation [%] | $T(t)$ during Simulation 3 | $T(t)$ during Simulation 4 |
|---|---|---|---|---|---|
| s1 | 30 | 29.59 | -1 | * | * |
| s2 | 28.72 | - | - | - | - |
| s3 | 19.1 | 27.97 | +46 | * | * |
| s4 | 18.8 | - | - | - | - |
| s5 | 15.32 | 26.56 | + 73 | * | * |
| s6 | 15.55 | - | - | - | - |

- no simulation conducted as due to the process settings and previous observations similar results are expected for: s1 and s2; s3 and s4; s5 and s6

* Good agreement, within temperature measurements

As the reason for deviations between experiment and simulation is detected, adaptions need to be made. Once the model provides good results and is validated the automated simulations can be used, for instance, to generate data for machine learning algorithms. Thereby, a huge amount of data can be generated with low effort. The automated simulation can be started within some minutes, as only some input variables and process parameter need to be defined. Further, practical experiments can be reduced to a great extent.

# 9. Conclusion and outlook

Four simulations, each representing a process step during the upsetting of preheated aluminum specimen, were developed. FE simulations were generated by using Abaqus and further, Abaqus Scripting Interface commands were used to set up Python scripts to automatically generate, run, and evaluate the simulations. A modular script structure was chosen, including one module for each simulation. Two additional modules were used to define necessary functions used by the simulation modules. Thereby, all four simulations are controlled by a main script, in which all process and simulation parameters are defined. By using a modular script structure, adaptions in the further development process will be easy to adapt, as each module, representing a simulation, can be modified and is exchangeable.

Sensor data, provided by the CPPSs during the experiments, was visualized and analyzed. Discrepancies between the preinstalled and the retrofitted thermocouple occurred, which need to be assessed. Varying furnace temperatures during heating lead to wider ranges in specimen temperature measured by the thermocouple. Changing the transport time from four to seven seconds results in a low temperature loss which has negligible impact on the upsetting force. However, temperature loss during the first seconds of transport seems to be significant. Measurements showed a strong temperature dependence of the specimen material. As expected, increasing the upset height also leads to higher forces.

Furthermore, literature research was conducted, to find reference values for material properties and other parameters used in the simulation. For the validation of the simulations a foundation was created, which allows to directly compare upsetting force and specimen temperature between simulation and experiments. Previous simulations were adapted and describe the temperature distribution of the specimen well. At lower specimen temperatures, the force predicted within the simulation is acceptable. However, the strong decrease in upsetting force caused by higher temperatures cannot be described with the model. It is assumed, that the Johnson-Cook parameter obtained from the literature do not accurately describe material behavior as parameters found in the literature show major differences, depending on the use of the model and the test setup. Therefore, experiments to determine Johnson-Cook parameter need to be conducted. Additionally, the specimen core temperature could be significantly higher, than predicted in the simulations, as the required upsetting force declines with higher temperatures. In this case, a more precise prediction of the temperature distribution inside the specimen in the simulations prior to the upsetting is necessary.

Furthermore, with reliable damage parameters the evaluation of the simulation can be extended to predict damage of the specimen. Therefore, damage parameters need to be calibrated with additional experiments.

The general approach was to create a rather detailed model, which can be further simplified once material and process parameters are adapted, and experiment and simulation show good correlation. A foundation was created to improve the process of comparing experiment and simulation. Further optimizations to reduce computation time are possible and can include, for instance, adaptions in the contact definition, mesh and mesh size, or the use of symmetry boundaries.

As an outlook, the automated simulation sequence can be further used to gather data for machine learning algorithms to make predictions about the model and to improve the process.

# List of Figures

**Figure 1:** Manufacturing processes [18] .......... 4
**Figure 2:** Dependence of the flow stress on the temperature [18] .......... 6
**Figure 3:** Upsetting of cylindrical part [18] .......... 7
**Figure 4:** Contact interaction (a) low pressure - contact through asperity peaks (b) moderate pressure - partial conformity (c) high pressure - full conformity [17] .......... 8
**Figure 5:** Friction models [17] .......... 9
**Figure 6:** Relevant process parameters [17] .......... 10
**Figure 7:** 1-D conduction through a wall [22] .......... 12
**Figure 8:** Convection [22] .......... 13
**Figure 9:** Radiation from a solid surface [22] .......... 13
**Figure 10:** Material models suitable for large deformations [20] .......... 19
**Figure 11:** Isotropic and kinematic hardening [18] .......... 20
**Figure 12:** Furnace at the Chair of Metal Forming .......... 21
**Figure 13:** Hydraulic press at the MF .......... 22
**Figure 14:** Schematic setup of the pyrometer position .......... 23
**Figure 15**: Example code .......... 26
**Figure 16**: Sensor data - load cell .......... 27
**Figure 17**: Sensor data - LVDT .......... 27
**Figure 18**: Sensor data - pyrometer .......... 28
**Figure 19**: Sensor data - pyrometer (detail) .......... 29
**Figure 20**: Sensor data - thermocouple furnace .......... 30
**Figure 21**: Sensor data - thermocouple .......... 30
**Figure 22**: Relevant time points from the pyrometer measurement .......... 31
**Figure 23:** Simulation 1 - Heating .......... 33
**Figure 24**: Simulation 3 – Rest on die .......... 35
**Figure 25:** Simulation 4 - Upsetting .......... 36
**Figure 26:** Partitions and mesh of the specimen .......... 37
**Figure 27:** Node set .......... 38
**Figure 28**: Interaction of Abaqus Scripting Interface commands with the Abaqus/CAE kernel [31] .... 44
**Figure 29**: Abaqus Output Database [31] .......... 46
**Figure 30**: Python script structure .......... 47
**Figure 31**: Overall process .......... 49
**Figure 32:** Automation of the simulation process .......... 51
**Figure 33**: Visualization of sensor data .......... 55
**Figure 34:** Temperatures (experiment 2, geometry A, $tt$ = 4 s, Δh = 5 mm) .......... 56
**Figure 35:** Upsetting force (experiment 2, geometry A, $tt$ = 4 s, Δh = 5 mm) .......... 56
**Figure 36:** Upsetting force (experiment 1, geometry A, $TF$ = 500 °C, $tt$ = 4 s / 7 s, Δh = 5 mm / 8 mm) .......... 57
**Figure 37:** Pyrometer temperature (experiment 1, geometry A, $TF$ = 500 °C, $tt$ = 4 s / 7 s, Δh = 5 mm / 8 mm) .......... 58
**Figure 38:** Comparison between experiments with the same test setting (s1): Load cell .......... 58
**Figure 39:** Comparison between experiments with the same test setting (s1): Pyrometer .......... 59
**Figure 40:** Comparison between experiments with the same test setting (s1): Thermocouple (furnace) .......... 60
**Figure 41:** Specimen after forming .......... 60
**Figure 42:** Heating curve - simulation 1 .......... 62

**Figure 43**: Specimen temperature after removing from the furnace .................................................. 63
**Figure 44:** Linear polynomial fit ........................................................................................................ 64
**Figure 45:** Reference values for the transport obtained from experiment 2 ...................................... 64
**Figure 46**: Influence of contact conductance........................................................................................ 65
**Figure 47:** Temperature distribution after a contact time of 8.5 s ...................................................... 66
**Figure 48:** Implicit vs. Explicit ............................................................................................................. 67
**Figure 49:** Influence of mesh size........................................................................................................ 67
**Figure 50**: Influence of different Johnson-Cook parameters ............................................................... 68
**Figure 51:** Upsetting simulation for setting s3 .................................................................................... 69
**Figure 52:** Comparison between experiment and simulation .............................................................. 71

# List of Tables

**Table 1:** Standardized designation of aluminum wrought alloys [21] ................................................ 11
**Table 2:** Chemical composition EN-AW-6082 [21] ................................................................ 11
**Table 3:** Sensor of the furnace [34] ................................................................................ 22
**Table 4:** Sensors of the hydraulic press [34] ..................................................................... 23
**Table 5:** Specimen dimensions ....................................................................................... 23
**Table 6:** General settings for all experiments ..................................................................... 24
**Table 7:** Experimental plan – experiment 1 ........................................................................ 25
**Table 8:** Experimental plan – experiment 2 ........................................................................ 25
**Table 9:** Unit systems ................................................................................................ 33
**Table 10**: Element type for each part ............................................................................. 37
**Table 11**: Thermal material properties of aluminum alloys [39] .............................................. 39
**Table 12**: Thermal material properties of silica [40] ........................................................... 39
**Table 13**: Thermal material properties of carbon steel [41] ................................................... 39
**Table 14**: Young's modulus for aluminum alloys [39] .......................................................... 40
**Table 15**: Young's modulus and Poisson ratio of steel [42] ................................................... 40
**Table 16:** Johnson Cook Model parameters ....................................................................... 41
T**able 17**: Johnson Cook damage parameters [44] ................................................................ 42
**Table 18**: Reference values - contact conductance ............................................................. 43
**Table 19**: Reverence values - convective heat transfer coefficient .......................................... 43
**Table 20:** Input and output parameter of each simulation ..................................................... 52
**Table 21:** Furnace temperatures ..................................................................................... 61
**Table 22:** Comparison between experiment 1 and simulation ................................................ 72
**Table 23:** Comparison between experiment 2 and simulation ................................................ 73

# References

[1] T. Zheng, M. Ardolino, A. Bacchetti, and M. Perona, “The applications of Industry 4.0 technologies in manufacturing context: a systematic literature review,” *International Journal of Production Research*, vol. 59, no. 6, pp. 1922–1954, 2021, doi: 10.1080/00207543.2020.1824085.

[2] Henning Kroll, Djerdj Horvat, Angela Jäger, “Effects of Automatisation and Digitalisation on Manufacturing Companies' Production Efficiency and Innovation Performance,” *Fraunhofer ISI Discussion Papers - Innovation Systems and Policy Analysis*, vol. 58. [Online]. Available: http://hdl.handle.net/10419/176701

[3] D. T. Matt, V. Modrák, and H. Zsifkovits, *Industry 4.0 for SMEs: Challenges, Opportunities and Requirements*. Cham: Springer International Publishing, 2020.

[4] Springer, Ed., *Simulation for Industry 4.0: Past, Present and Future*. Cham: Springer International Publishing, 2019.

[5] J. N. Reddy, *An introduction to nonlinear finite element analysis: with applications to heat transfer, fluid mechanics, and solid mechanics*. Oxford: Oxford University Press, 2015.

[6] B. Rodič, “Industry 4.0 and the New Simulation Modelling Paradigm,” *Organizacija*, vol. 50, no. 3, pp. 193–207, 2017, doi: 10.1515/orga-2017-0017.

[7] S.Y. Lin, “Upsetting of a cylindrical specimen between elastic tools,” *Journal of Materials Processing Technology*, vol. 86, no. 1, pp. 73–80, 1999, doi: 10.1016/S0924-0136(98)00236-2.

[8] P. Tuğcu, “Thermomechanical analysis of upsetting of a cylindrical billet,” *Computers & Structures*, vol. 58, no. 1, pp. 1–12, 1996, doi: 10.1016/0045-7949(95)00122-W.

[9] M. V. Murashov and A. V. Vlasov, “Three-dimensional Finite Element Modelling of the Cylindrical Specimen Upsetting,” *MATEC Web of Conferences*, vol. 220, p. 4008, 2018, doi: 10.1051/matecconf/201822004008.

[10] Chun-Ho Liu, A-Cheng Wang, Yi-Sian Chen, and Chien-Ming Wang, “The coupled thermo-mechanical analysis in the upsetting process by the dynamic FEM,” *Journal of Materials Processing Technology*, vol. 201, no. 1, pp. 37–42, 2008, doi: 10.1016/j.jmatprotec.2007.11.174.

[11] Z.J. Zhang, G.Z. Dai, S.N. Wu, L.X. Dong, and L.L. Liu, “Simulation of 42CrMo steel billet upsetting and its defects analyses during forming process based on the software DEFORM-3D,” *Materials Science and Engineering: A*, vol. 499, no. 1, pp. 49–52, 2009, doi: 10.1016/j.msea.2007.11.135.

[12] M. Nytra, P. Kubík, J. Petruška, and F. Šebek, “A Fully Coupled Thermomechanical Damage Analysis of Hot Closed Die Forging Using Finite Element Modeling,” *Journal of Materials Engineering and Performance*, vol. 29, no. 12, pp. 8236–8246, 2020, doi: 10.1007/s11665-020-05252-4.

[13] Siamak Serajzadeh, “Prediction of thermo-mechanical behavior during hot upsetting using neural networks,” *Materials Science and Engineering: A*, vol. 472, no. 1, pp. 140–147, 2008, doi: 10.1016/j.msea.2007.03.037.

[14] Yong-Cheng Lin and Ming-Song Chen, “Numerical simulation and experimental verification of microstructure evolution in a three-dimensional hot upsetting process,” *Journal of Materials Processing Technology*, vol. 209, no. 9, pp. 4578–4583, 2009, doi: 10.1016/j.jmatprotec.2008.10.036.

[15] Roland Rosen, Georg von Wichert, George Lo, and Kurt D. Bettenhausen, “About The Importance of Autonomy and Digital Twins for the Future of Manufacturing,” *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 567–572, 2015, doi: 10.1016/j.ifacol.2015.06.141.

[16] H. Tschaetsch, *Metal forming practise: Processes - machines - tools*. Berlin, New York: Springer-Verlag, 2006.

[17] H. S. Valberg, *Applied Metal Forming: Including FEM Analysis: including FEM analysis*. New York: Cambridge University Press, 2010.

[18] E. Doege and B.-A. Behrens, *Handbuch Umformtechnik: Grundlagen, Technologien, Maschinen,* 3rd ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[19] *Dictionary of Production Engineering I / Wörterbuch der Fertigungstechnik I / Dizionario di Ingegneria della Produzione I: Metal Forming / Umformtechnik / Formatura dei Metalli*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019.

[20] F. Klocke, *Fertigungsverfahren 4*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017.

[21] F. Ostermann, *Anwendungstechnologie Aluminium,* 3rd ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. [Online]. Available: http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1568831

[22] J.-C. Han, *Analytical heat transfer*. Boca Raton, FL: CRC Press, 2012. [Online]. Available: https://www.taylorfrancis.com/books/9780429109652

[23] T. L. Bergman and A. S. Lavine, *Fundamentals of heat and mass transfer*. Hoboken, NJ: John Wiley & Sons, 2017. [Online]. Available: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6488272

[24] Ralph Benjamin James and Martin Stockinger, “Digitalization and digital transformation in metal forming: key technologies challenges and current developments of industry 4.0 applications,” in *XXXIX. Verformungskundliches Kolloquium*, 2020, pp. 13–23.

[25] J. Pistorius, *Industrie 4. 0 - Schlüsseltechnologien Für Die Produktion: Grundlagen * Potenziale * Anwendungen*. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2020. [Online]. Available: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6245733

[26] H. K. Tönshoff and I. Inasaki, Eds., *Sensors in manufacturing*. Weinheim, Cambridge: Wiley-VCH, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/3527600027

[27] X. Wu, V. Goepp, and A. Siadat, “Concept and engineering development of cyber physical production systems: a systematic literature review,” *The International Journal of Advanced Manufacturing Technology*, vol. 111, no. 1, pp. 243–261, 2020, doi: 10.1007/s00170-020-06110-2.

[28] M. Sorger, B. J. Ralph, K. Hartl, M. Woschank, and M. Stockinger, “Big Data in the Metal Processing Value Chain: A Systematic Digitalization Approach under Special Consideration of Standardization and SMEs,” *Applied Sciences*, vol. 11, no. 19, pp. 1–22, 2021, doi: 10.3390/app11199021.

[29] P. Osterrieder, L. Budde, and T. Friedli, “The smart factory as a key construct of industry 4.0: A systematic literature review,” *International Journal of Production Economics*, vol. 221, p. 107476, 2020, doi: 10.1016/j.ijpe.2019.08.011.

[30] D. W. Pepper and J. C. Heinrich, *The finite element method: Basic concepts and applications,* 2nd ed. New York: Taylor & Francis, 2006. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0653/2005002971-d.html

[31] Dassault Systems, *Abaqus Documentation*.

[32] A. Begovic, “FE-Parameterstudie eines thermomechanisch gekoppelten Stauchversuches in Abaqus,” Lehrstuhl für Umformtechnik, Montanuniversität, Leoben, 2021.

[33] G. Li, J. Jinn, W. Wu, and S. Oh, “Recent development and application of three-dimensional finite element modelling in bulk forming process,” *Journal of Materials Processing Technology*, vol. 113, pp. 40–45, 2001.

[34] B. J. Ralph *et al.,* “MUL 4.0: Systematic Digitalization of a Value Chain from Raw Material to Recycling,” *Procedia Manufacturing*, vol. 55, pp. 335–342, 2021, doi: 10.1016/j.promfg.2021.10.047.

[35] M. Woschank *et al.,* “MUL 4.0 – Digitalisierung der Wertschöpfungskette vom Rohmaterial bis hin zum Recycling,” *Berg Huettenmaenn Monatsh*, vol. 166, no. 6, pp. 309–313, 2021, doi: 10.1007/s00501-021-01119-w.

[36] Wikipedia, *Load cell.* [Online]. Available: https://en.wikipedia.org/w/index.php?title=Load_cell&oldid=1085892888 (accessed: Aug. 8 2022).

[37] *Grundlagen der Infrarot-Temperaturmessung.* [Online]. Available: https://www.keller.de/de/its/pyrometer/applikationen/grundlagen/grundlagen-der-infrarot-temperaturmessung.htm (accessed: Aug. 8 2022).

[38] *Thermocouple probes.* [Online]. Available: https://www.omega.com/en-us/resources/thermocouple-hub (accessed: Aug. 8 2022).

[39] *EN 1999-1-2: Eurocode 9: Design of aluminium structures - Part 1-2: Structural fire design*, 1999-1-2, 2007.

[40] A. Eschner, *Thermophysikalische Stoffwerte von feuerfesten Materialien - PDF Kostenfreier Download.* [Online]. Available: https://docplayer.org/134801229-Thermophysikalische-stoffwerte-von-feuerfesten-materialien.html (accessed: Feb. 12 2022).

[41] “Annex A: Thermal Data for Carbon Steel and Stainless Steel Sections,” [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9783433601570.app1

[42] C.-H. Yeh, N. Jeyaprakash, and C.-H. Yang, “Non-destructive characterization of elastic properties on steel plate using laser ultrasound technique under high-temperature atmosphere,” *The International Journal of Advanced Manufacturing Technology*, vol. 108, 1-2, pp. 129–141, 2020, doi: 10.1007/s00170-020-05383-x.

[43] B. J. Ralph, K. Hartl, M. Sorger, A. Schwarz-Gsaxner, and M. Stockinger, “Machine Learning Driven Prediction of Residual Stresses for the Shot Peening Process Using a Finite Element Based Grey-Box Model Approach,” *JMMP*, vol. 5, no. 2, p. 39, 2021, doi: 10.3390/jmmp5020039.

[44] A. Rusinek, A. Arias, M. Rodríguez Millán, and D. Garcia Gonzalez, “Influence of Stress State on the Mechanical Impact and Deformation Behaviors of Aluminum Alloys,” *Metals*, vol. 8, 2018, doi: 10.3390/met8070520.

[45] Y. Akif, “Investıgatıon of Cold Weldıng At Steel-Alumınum Combınatıons Vıa Extrusıon Process Usıng Thermo-Mechanıcally Coupled Fınıte Element Analysıs,” Master Thesis, Graduate School of Science and Engineering of Hacettepe University, 2019.

[46] X. Chen, Y. Peng, S. Peng, S. Yao, C. Chen, and P. Xu, “Flow and fracture behavior of aluminum alloy 6082-T6 at different tensile strain rates and triaxialities,” *PLOS ONE*, vol. 12, no. 7, e0181983, 2017, doi: 10.1371/journal.pone.0181983.

[47] Bowen Liang, Ronghua Li, Xiujuan Zhang, and Yanjun Chen, “Finite Element Analysis of Friction Stir Welded 6082-T6 Aluminum Alloy’s Residual Stress,” *Journal of Physics: Conference Series*, vol. 1605, 2020, doi: 10.1088/1742-6596/1605/1/012121.

[48] Peng Yibo, Wang Gang, Zhu Tianxing, Pan Shangfeng, and Rong Yiming, “Dynamic Mechanical Behaviors of 6082-T6 Aluminum Alloy,” *Advances in Mechanical Engineering*, vol. 5, pp. 1–8, 2013, doi: 10.1155/2013/878016.

[49] J. Ning and S. Y. Liang, “Inverse identification of Johnson-Cook material constants based on modified chip formation model and iterative gradient search using temperature and force measurements,” *The International Journal of Advanced Manufacturing Technology*, vol. 102, 9-12, 2019, doi: 10.1007/s00170-019-03286-0.

[50] J.-L. Pérez-Castellanos and A. Rusinek, “Temperature increase associated with plastic deformation under dynamic compression: application to aluminum alloy Al 6082,” *Journal of theoretical and applied mechanics*, vol. 50, pp. 377–398, 2012.

[51] M Rosochowska, R Balendra, and K Chodnikiewicz, “Measurements of thermal contact conductance,” *Journal of Materials Processing Technology*, vol. 135, no. 2, pp. 204–210, 2003, doi: 10.1016/S0924-0136(02)00897-X.

[52] Y. Chang, X. Tang, K. Zhao, P. Hu, and Y. Wu, *Investigation of the factors influencing the interfacial heat transfer coefficient in hot stamping*, 2014.

[53] L. Ying, T. Gao, M. Dai, and P. Hu, “Investigation of interfacial heat transfer mechanism for 7075-T6 aluminum alloy in HFQ hot forming process,” *Applied Thermal Engineering*, vol. 118, pp. 266–282, 2017, doi: 10.1016/j.applthermaleng.2017.02.107.

[54] J.E. Akin, “FEA Concepts: SW Simulation Overview,” [Online]. Available: https://www.clear.rice.edu/mech403/HelpFiles/FEA_thermal_concepts.pdf

[55] Q. Bai, J. Lin, L. Zhan, T. A. Dean, D. S. Balint, and Z. Zhang, “An efficient closed-form method for determining interfacial heat transfer coefficient in metal forming,” *International Journal of Machine Tools and Manufacture*, vol. 56, pp. 102–110, 2012, doi: 10.1016/j.ijmachtools.2011.12.005.

[56] R. Marek and K. Nitsche, Eds., *Praxis der Wärmeübertragung*. München: Carl Hanser Verlag GmbH & Co. KG, 2015.

[57] Martin Pletz, “Efficient Finite Element Modelling: Automated model generation & evaluation using Simulia Abaqus,” Chair of Desinging Plastics and Composite Materials, Montanuniversität Leoben, Austria, 2021.

# Appendix

## Appendix A: measurements.py

```python
# -------------------------------------------- SCRIPT INFORMATION ----------------------------------------------

# script name: measurements_21062022
# function: evaluate and visualizes the sensor data
# script includes the experiment information (section FILEPATH / EXPERIMENTS) for experiment 1

# -------------------------------------------- IMPORT ----------------------------------------------------------

import matplotlib.pyplot as plt
import numpy as np
import os
from datetime import timedelta

# -------------------------------------------- FILE PATHS ------------------------------------------------------

MAIN_DIR = r'L:\090_Datenaustausch\cwaiguny\MA\project_21062022'
MEASUREMENT_FOLDER = os.path.join(MAIN_DIR, r'experiment\sensor')
OUTPUT_FOLDER = os.path.join(MAIN_DIR, r'results')
SAVE = True  # if true, the figures are saved

# -------------------------------------------- EXPERIMENTS -----------------------------------------------------
# 1.) define parameter for each test setting: setting name/number, tested geometry ('A', 'B'), furnace temperature [°C],
# upsetting height [mm], transport time [s], rest time on bottom die [s]
s_1 = ['s1', 10, 15, 300, 5, 4, 3]
s_2 = ['s2', 10, 15, 300, 8, 4, 3]
s_3 = ['s3', 20, 30, 300, 15, 4, 3]
s_4 = ['s4', 20, 30, 300, 20, 7, 3]
s_5 = ['s5', 10, 15, 500, 5, 7, 3]
s_6 = ['s_6', 10, 15, 500, 8, 4, 3]
s_7 = ['s7', 20, 30, 500, 15, 7, 3]
s_8 = ['s8', 20, 30, 500, 20, 4, 3]
settings = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]  # all test settings

# 2.) enter the test number of the measurement that correspond to the test setting
sp_1 = ['TestNr_4.', 'TestNr_8.', 'TestNr_9.', 'TestNr_10.', 'TestNr_11.', 'TestNr_12.']
sp_2 = ['TestNr_13.', 'TestNr_14.', 'TestNr_15.', 'TestNr_16.', 'TestNr_17.']
sp_3 = ['TestNr_19.', 'TestNr_20.', 'TestNr_21.', 'TestNr_22.', 'TestNr_23.', 'TestNr_24.']
sp_4 = ['TestNr_31.', 'TestNr_26.', 'TestNr_27.', 'TestNr_32.', 'TestNr_29.', 'TestNr_30.']
sp_5 = ['TestNr_33.', 'TestNr_34.', 'TestNr_35.', 'TestNr_36.', 'TestNr_37.', 'TestNr_38.']
sp_6 = ['TestNr_39.', 'TestNr_40.', 'TestNr_41.', 'TestNr_42.', 'TestNr_43.', 'TestNr_44.']
sp_7 = ['TestNr_45.', 'TestNr_46.', 'TestNr_47.', 'TestNr_48.', 'TestNr_49.', 'TestNr_50.']
sp_8 = ['TestNr_51.', 'TestNr_52.', 'TestNr_53.', 'TestNr_54.', 'TestNr_55.', 'TestNr_56.']
specimen = [sp_1, sp_2, sp_3, sp_4, sp_5, sp_6, sp_7, sp_8]  # corresponding measurement numbers


# -------------------------------------------- FUNCTIONS -------------------------------------------------------
def get_seconds(time_string):
    t = time_string.split('#')[1]
    if 'ms' in t:
        t = t.strip('ms')
        if 'm' in t and 's' in t:
            t = t.split('m')
            t1 = t[1].split('s')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t1[0]), milliseconds=int(t1[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 's' in t:
            t = t.split('s')
            time_ = timedelta(seconds=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        else:
            time_ = timedelta(milliseconds=int(t))
            time_ = time_.total_seconds()
    else:
        if 'm' in t and 's' in t:
            t = t.strip('s')
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.strip('m')
            time_ = timedelta(minutes=int(t))
            time_ = time_.total_seconds()
        elif 's':
            time_ = int(t.strip('s'))
    return time_


def read_measurement(filename_):
    # this function depends on structure of the measurement file
    time_ = []
    load_ = []
    abs_gap_ = []
    rel_gap_ = []
    t_pyro_ = []
    t_thermo_ = []
    t_left_ = []
    with open(filename_, 'r') as f:
        header_ = f.readline()
        for line in f:
            d = line.strip().split(';')
            time_.append(get_seconds(d[0]))
            load_.append(float(d[1]))
            abs_gap_.append(float(d[2]))
            rel_gap_.append(float(d[3]))
            t_pyro_.append(float(d[4]))
            t_thermo_.append(float(d[5]))
            t_left_.append(float(d[6]))
    return header_, time_, load_, abs_gap_, rel_gap_, t_pyro_, t_thermo_, t_left_


def plot_layout(settings_, m):
    fig = plt.figure(figsize=(15, 10))
    sub1_ = fig.add_subplot(231)
    sub1_.set_title('Load cell')
    sub1_.set_ylabel('load [kN]')
    sub1_.set_xlabel('time [s]')

    sub2_ = fig.add_subplot(232)
    sub2_.set_title('LVDT sensor')
    sub2_.set_ylabel('absolute gap [mm]')
    sub2_.set_xlabel('time [s]')

    sub3_ = fig.add_subplot(234)
    sub3_.set_title('Pyrometer')
    sub3_.set_ylabel('temperature [°C]')
    sub3_.set_xlabel('time [s]')

    sub4_ = fig.add_subplot(235)
    sub4_.set_title('Thermocouple')
    sub4_.set_ylabel('temperature [°C]')
    sub4_.set_xlabel('time [s]')

    sub5_ = fig.add_subplot(236)
    sub5_.set_title('Thermocouple')
    sub5_.set_ylabel('temperature [°C]')
    sub5_.set_xlabel('time [s]')

    title = ('MEASUREMENTS \n' + settings_[m][0] + ': specimen geometry: d0 = ' +
             str(settings_[m][1]) + ' [mm] h0= ' + str(settings_[m][2]) + ' [mm], T= ' +
             str(settings_[m][3]) + ' [°C], delta h = ' + str(settings_[m][4]) +
             ' [mm], transfer time = ' + str(settings_[m][5]) + ' [s], rest time = ' +
             str(settings_[m][6]) + '[s]')
    plt.suptitle(title)
    return sub1_, sub2_, sub3_, sub4_, sub5_


def calculate_velocity(abs_gap_, time_, h):
    start = 0
    for ind, gap in enumerate(abs_gap_):
        if gap < h:
            start = ind
            break
    minimum = min(abs_gap_)
    end = abs_gap_.index(minimum)
    s = abs_gap_[start] - abs_gap_[end]
    v_ = s / (time_[end] - time_[start])
    return v_


# ----------------------------------------------- CALCULATIONS -------------------------------------------------------
plt.close('all')

files = os.listdir(MEASUREMENT_FOLDER)  # list all measurement
nr = len(settings)
for i in range(nr):

    v = []
    t_s = []
    t_f = []
    F_max = []
    sub1, sub2, sub3, sub4, sub5 = plot_layout(settings, i)

    for j in range(len(specimen[i])):  # evaluate and plot measurements
        for file in files:
            if specimen[i][j] in file:  # search for the test setting
                filepath = os.path.join(MEASUREMENT_FOLDER, file)
                header, time, load, abs_gap, rel_gap, t_pyro, t_thermo, t_left = read_measurement(filepath)

                for number, elem in enumerate(load):
                    # correction factor of force data (changes with sensor signal)
                    load[number] = elem * 1.27

                F_max.append(np.max(load))  # maximum force

                # average velocity
                abs_pos = abs_gap[0]  # absolute position of the top die at the start
                v.append(calculate_velocity(abs_gap, time, abs_pos - 1))

                t_f.append(np.mean(t_left))  # average furnace temperature

                t_s.append(np.mean(t_thermo))  # average temperature of the thermocouple

                filename = file.strip('.csv')
                sub1.plot(time, load, label=filename)
                sub2.plot(time, abs_gap, label=filename)
                sub3.plot(time, t_pyro, label=filename)
                sub4.plot(time, t_left, label=filename)
                sub5.plot(time, t_thermo, label=filename)

    sub2.legend(loc='upper right', bbox_to_anchor=(1.8, 1.02))
    if SAVE:
        name = 'sensor_data_' + settings[i][0]
        name = os.path.join(OUTPUT_FOLDER, name)
        plt.savefig(name, dpi=600)

    print(settings[i][0])
    velocity = np.mean(v)
    print('average velocity of hydraulic press:', "{:.1f}".format(velocity), ' mm/s')
    temp_surrounding = np.mean(t_s)
    print('average temperature thermocouple: ', "{:.0f}".format(temp_surrounding), ' °C')
    t_furnace = np.mean(t_f)
    print('average temperature thermocouple in furnace: ', "{:.0f}".format(t_furnace), ' °C')
    average_max_force = np.mean(F_max)
    print('average maximum force: ', "{:.2f}".format(average_max_force), ' kN')

plt.show()
# -----------------------------------------------------------------------------------------------------------

```

```python
# ------------------------------------------- SCRIPT INFORMATION ---------------------------------------------------

# script name: measurements_15072022
# function: evaluate and visualizes the sensor data
# script includes the experiment information (section FILEPATH / EXPERIMENTS) for experiment 2

# ------------------------------------------- IMPORT ------------------------------------------------------------

import matplotlib.pyplot as plt
import numpy as np
import os
from datetime import timedelta

# ------------------------------------------- FILE PATHS ---------------------------------------------------------

MAIN_DIR = r'L:\090_Datenaustausch\cwaiguny\MA\project_15072022'
MEASUREMENT_FOLDER = os.path.join(MAIN_DIR, r'experiment\sensor')
OUTPUT_FOLDER = os.path.join(MAIN_DIR, r'results')
SAVE = True  # if true, the figures are saved

# ------------------------------------------- EXPERIMENTS ------------------------------------------------------
# 1.) define parameter for each test setting: setting name/number, tested geometry ('A', 'B'), furnace
temperature [°C],
# upsetting height [mm], transport time [s], rest time on bottom die [s]
s_1 = ['s1', 10, 15, 300, 5, 4, 3]
s_2 = ['s2', 10, 15, 300, 5, 7, 3]
s_3 = ['s3', 10, 15, 400, 5, 4, 3]
s_4 = ['s4', 10, 15, 400, 5, 7, 3]
s_5 = ['s5', 10, 15, 500, 5, 4, 3]
s_6 = ['s6', 10, 15, 500, 5, 7, 3]
settings = [s_1, s_2, s_3, s_4, s_5, s_6]  # all test settings

# 2.) enter the test number of the measurement that correspond to the test setting
sp_1 = ['TestNr_12.', 'TestNr_13.', 'TestNr_14.', 'TestNr_15.']
sp_2 = ['TestNr_16.', 'TestNr_17.', 'TestNr_18.', 'TestNr_19.']
sp_3 = ['TestNr_20.', 'TestNr_21.', 'TestNr_22.', 'TestNr_23.']
sp_4 = ['TestNr_24.', 'TestNr_25.', 'TestNr_26.', 'TestNr_27.']
sp_5 = ['TestNr_28.', 'TestNr_29.', 'TestNr_30.', 'TestNr_31.']
sp_6 = ['TestNr_32.', 'TestNr_33.', 'TestNr_34.', 'TestNr_35.']
specimen = [sp_1, sp_2, sp_3, sp_4, sp_5, sp_6]

# ------------------------------------------- FUNCTIONS ---------------------------------------------------------


def get_seconds(time_string):
    t = time_string.split('#')[1]
    if 'ms' in t:
        t = t.strip('ms')
        if 'm' in t and 's' in t:
            t = t.split('m')
            t1 = t[1].split('s')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t1[0]), milliseconds=int(t1[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 's' in t:
            t = t.split('s')
            time_ = timedelta(seconds=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        else:
            time_ = timedelta(milliseconds=int(t))
            time_ = time_.total_seconds()
    else:
        if 'm' in t and 's' in t:
            t = t.strip('s')
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.strip('m')
            time_ = timedelta(minutes=int(t))
            time_ = time_.total_seconds()
        elif 's':
            time_ = int(t.strip('s'))
    return time_


def read_measurement(filename_):
    # this function depends on structure of the measurement file
    time_ = []
    load_ = []
    abs_gap_ = []
    rel_gap_ = []
    t_pyro_ = []
    t_thermo_ = []
    t_left_ = []
    with open(filename_, 'r') as f:
        header_ = f.readline()
        for line in f:
            d = line.strip().split(';')
            time_.append(get_seconds(d[0]))
            load_.append(float(d[1]))
            abs_gap_.append(float(d[2]))
            rel_gap_.append(float(d[3]))
            t_pyro_.append(float(d[4]))
            t_thermo_.append(float(d[6]))
            t_left_.append(float(d[7]))
    return header_, time_, load_, abs_gap_, rel_gap_, t_pyro_, t_thermo_, t_left_


def plot_layout(settings_, m):
    fig = plt.figure(figsize=(30, 20))
    sub1_ = fig.add_subplot(231)
    sub1_.set_title('Load cell')
    sub1_.set_ylabel('load [kN]')
    sub1_.set_xlabel('time [s]')

    sub2_ = fig.add_subplot(232)
    sub2_.set_title('LVDT')
    sub2_.set_ylabel('absolute gap [mm]')
    sub2_.set_xlabel('time [s]')

    sub3_ = fig.add_subplot(234)
    sub3_.set_title('Pyrometer')
```

```python
    sub3_.set_ylabel('temperature [°C]')
    sub3_.set_xlabel('time [s]')

    sub4_ = fig.add_subplot(235)
    sub4_.set_title('Thermocouple furnace')
    sub4_.set_ylabel('temperature [°C]')
    sub4_.set_xlabel('time [s]')

    sub5_ = fig.add_subplot(236)
    sub5_.set_title('Thermocouple')
    sub5_.set_ylabel('temperature [°C]')
    sub5_.set_xlabel('time [s]')

    title = ('MEASUREMENTS \n' + settings_[m][0] + ': specimen geometry: d0 = ' +
             str(settings_[m][1]) + ' [mm] h0= ' + str(settings_[m][2]) + ' [mm], T= ' +
             str(settings_[m][3]) + ' [°C], delta h = ' + str(settings_[m][4]) +
             ' [mm], transfer time = ' + str(settings_[m][5]) + ' [s], rest time = ' +
             str(settings_[m][6]) + '[s]')
    plt.suptitle(title)
    return sub1_, sub2_, sub3_, sub4_, sub5_


def calculate_velocity(abs_gap_, time_, h):
    start = 0
    for ind, gap in enumerate(abs_gap_):
        if gap < h:
            start = ind
            break
    minimum = min(abs_gap_)
    end = abs_gap_.index(minimum)
    s = abs_gap_[start] - abs_gap_[end]
    v_ = s / (time_[end] - time_[start])
    return v_


# ------------------------------------------------ CALCULATIONS -----------------------------------------------------------
plt.close('all')

files = os.listdir(MEASUREMENT_FOLDER)  # list all measurement
nr = len(settings)
for i in range(nr):

    v = []
    t_s = []
    t_f = []
    F_max = []
    sub1, sub2, sub3, sub4, sub5 = plot_layout(settings, i)
    plt.style.use('seaborn-ticks')

    for j in range(len(specimen[i])):  # evaluate and plot measurements
        for file in files:
            if specimen[i][j] in file:  # search for the test setting
                filepath = os.path.join(MEASUREMENT_FOLDER, file)
                header, time, load, abs_gap, rel_gap, t_pyro, t_thermo, t_left = read_measurement(filepath)

                F_max.append(np.max(load))  # maximum force

                # average velocity
                abs_pos = abs_gap[0]  # absolute position of the top die at the start
                v.append(calculate_velocity(abs_gap, time, abs_pos - 1))

                t_f.append(np.mean(t_left))  # average furnace temperature

                t_s.append(np.mean(t_thermo))  # average temperature of the thermocouple

                filename = file.strip('.csv')
                sub1.plot(time, load, label=filename)
                sub2.plot(time, abs_gap, label=filename)
                sub3.plot(time, t_pyro, label=filename)
                sub4.plot(time, t_left, label=filename)
                sub5.plot(time, t_thermo, label=filename)

    sub2.legend(loc='upper right', bbox_to_anchor=(1.8, 1.02))
    if SAVE:
        name = 'sensor_data_' + settings[i][0]
        name = os.path.join(OUTPUT_FOLDER, name)
        plt.savefig(name, dpi=600)

    print(settings[i][0])
    velocity = np.mean(v)
    print('average velocity of hydraulic press:', "{:.1f}".format(velocity), ' mm/s')
    temp_surrounding = np.mean(t_s)
    print('average temperature thermocouple: ', "{:.0f}".format(temp_surrounding), ' °C')
    t_furnace = np.mean(t_f)
    print('average temperature thermocouple in furnace: ', "{:.0f}".format(t_furnace), ' °C')
    average_max_force = np.mean(F_max)
    print('average maximum force: ', "{:.2f}".format(average_max_force), ' kN')

plt.show()
# ---------------------------------------------------------------------------------------------

```

## Appendix B: main_script.py

```
# ------------------------------------------ SCRIPT INFORMATION -------------------------------------------------
#
# name: main_script
# function: generate, run and evaluate simulations
# info: change abaqus working directory to the path of this script before running this script
# unit system: SI-mm and temperatures in degree Celcius
#
# ------------------------------------------ IMPORT -------------------------------------------------------------

from abaqus import *
from abaqusConstants import *
from caeModules import *
import os
from modules import simulation_1
from modules import simulation_2p
from modules import simulation_3p
from modules import simulation_4i
from modules import simulation_4e
from modules import odb_data

# ------------------------------------------ FILE PATHS ---------------------------------------------------------

SCRIPT_PATH = os.getcwd()
MAIN_DIR = r'L:\090_Datenaustausch\cwaiguny\MA\project_21062022'
RESULT_PATH = r'L:\090_Datenaustausch\cwaiguny\MA\project_21062022\sim\res'
CSV_PATH = r'L:\090_Datenaustausch\cwaiguny\MA\project_21062022\sim\csv'

# ------------------------------------------ SPECIMEN -----------------------------------------------------------

setting = 's1_2'  # test setting name / identification number
d0 = 10  # initial diameter of specimen
h0 = 15  # initial height of the specimen
h1 = 10   # height of the specimen after forming

# ------------------------------------------ PROCESS PARAMETER -------------------------------------------------

ambient_temperature = 28       # ambient temperature
furnace_temperature = 276      # temperature inside the furnace (preheating temperature)

heating_time = 1200           # process time of the specimen in the furnace
transport_time = 4           # transport time from furnace to the hydraulic press
rest_time = 3              # rest time of specimen on bottom die before the upsetting process starts

v = 6.3            # velocity of hydraulic press
hi = 50            # initial distance between top and bottom die

moving_time = (hi - h0) / v          # time to move the top die onto the top surface of the specimen
contact_time = moving_time + rest_time  # total contact time for heat transfer to bottom die
delta_h = h0 - h1                  # upsetting height
upsetting_time = delta_h / v         # time for the upsetting process

temperature_amplitude = ((0.0, furnace_temperature), (heating_time, furnace_temperature),)  # 
temperature in the furnace
amplitude_displacement_data = ((0.0, 0.0), (upsetting_time, delta_h))  # time - displacement amplitude 
of the top die

# ------------------------------------------ SIMULATION PARAMETER -----------------------------------------------

```

```
emissivity = 0.3       # emissivity of the specimen
emissivity_f = 0.8     # emissivity of the furnace lining

# heat transfer coefficient for convectional heat transfer
convection_coeff_1 = 0.025  # free convection in the furnace
convection_coeff_2 = 0.15  # forced convection during transport
convection_coeff_3 = 0.025  # free convection during rest on die

# contact conductance as a function of clearance
contact_conductance_1 = ((0.3, 0.0), (0.0, 0.01))  # between furnace and specimen
contact_conductance_3 = ((2, 0.0), (0.0, 0.01))   # between bottom die and specimen
contact_conductance_4 = ((20, 0.0), (0.0, 0.01))   # between dies and specimen during upsetting

friction = 0.3          # friction coefficient in contact area between specimen and bottom die

seed_size = 0.2            # global seed size for the specimen
seed_size_furnace = 10     # global seed size for the furnace
seed_size_die = 1          # global seed size for the dies

# -------------------------------------------- FILE NAMES ----------------------------------------------------------

# naming of FE simulations
name_1 = 'sim1_' + str(setting)
name_3 = 'sim3_' + str(setting)
name_4i = 'sim4i_' + str(setting)
name_4e = 'sim4e_' + str(setting)

# file paths to odb-files of previous simulation
odb_abs_path_2 = os.path.join(RESULT_PATH, name_1 + '.odb')
odb_abs_path_4 = os.path.join(RESULT_PATH, name_3 + '.odb')

# --------------------------------------------------------------------------------------------------------------------

# set variable to 'True' to define which simulations to execute
# note: simulation results from previous simulation are needed
runSim1 = True
runSim2p_3p = True
runSim4i = True
runSim4e = False

# -------------------------------------------- SIMULATION 1 ----------------------------------------------------------

if runSim1:
   simulation_1.heating(h0, d0, ambient_temperature, furnace_temperature, heating_time, emissivity
, emissivity_f,
                      convection_coeff_1, temperature_amplitude, contact_conductance_1, seed_size,
                      seed_size_furnace, RESULT_PATH, name_1)

   T1_end = odb_data.evaluate_end_temperature(name_1)

# -------------------------------------------- SIMULATION 2 + SIMULATION 3  ------------------------------------------

if runSim2p_3p:
   T2 = T1_end
   T2_end = simulation_2p.transport(d0, h0, ambient_temperature, T2, transport_time, emissivity,
convection_coeff_2)

   T3 = T2_end
```

```
    simulation_3p.rest(h0, d0, contact_time, emissivity, ambient_temperature, T3, contact_conductance_3,
                       convection_coeff_3, seed_size, seed_size_die, RESULT_PATH, name_3)

    odb_data.evaluate_temperature(name_3, CSV_PATH)


# ------------------------------------------- SIMULATION 4 - IMPLICIT ----------------------------------------------

if runSim4i:
    simulation_4i.upsetting(h0, d0, ambient_temperature, upsetting_time, friction, contact_conductance_4,
                            odb_abs_path_4, amplitude_displacement_data, seed_size, seed_size_die, RESULT_PATH, name_4i)

    odb_data.evaluate_upsetting(name_4i, CSV_PATH)

# ------------------------------------------- SIMULATION 4 - EXPLICIT ----------------------------------------------

if runSim4e:
    simulation_4e.upsetting(h0, d0, ambient_temperature, upsetting_time, friction, contact_conductance_4,
                            odb_abs_path_4, amplitude_displacement_data, seed_size, seed_size_die, RESULT_PATH, name_4e)

    odb_data.evaluate_upsetting(name_4e, CSV_PATH)

# ------------------------------------------------------------------------------------------------------------------

```

## Appendix C: compare.py

```python
# ------------------------------------------------ SCRIPT INFORMATION -------------------------------------------------------
#
# name: compare_21062022
# function: compare experiment and simulation
# script includes the experiment information (section FILEPATH / EXPERIMENTS) for experiment 1

# ------------------------------------------------ IMPORT -------------------------------------------------------------

import matplotlib.pyplot as plt
import numpy as np
import os
from datetime import timedelta
from lmfit.models import LinearModel

# ------------------------------------------------ FILE PATHS ------------------------------------------------------------

MAIN_DIR = r'L:\090_Datenaustausch\cwaiguny\MA\project_21062022'
MEASUREMENT_FOLDER = os.path.join(MAIN_DIR, r'experiment\sensor')
SIMULATION_RESULTS_FOLDER = os.path.join(MAIN_DIR, r'sim\csv')
OUTPUT_FOLDER = os.path.join(MAIN_DIR, r'results')

SAVE = True  # if true, the figures are saved
showFit = False  # include linear fit in the plot

# ------------------------------------------------ PARAMETER -----------------------------------------------------------

hp = 6.0  # pyrometer position [mm]
tol = 0.5  # tolerance for pyrometer position [mm]

# ------------------------------------------------ EXPERIMENTS --------------------------------------------------------
# 1.) define parameter for each test setting: setting name/number, initial diameter d0 [mm], initial
height h0 [mm],
# furnace temperature [°C], upsetting height [mm], transport time [s], rest time on bottom die [s]

s_1 = ['s1', 10, 15, 300, 5, 4, 3]
s_2 = ['s2', 10, 15, 300, 8, 4, 3]
s_3 = ['s3', 20, 30, 300, 15, 4, 3]
s_4 = ['s4', 20, 30, 300, 20, 7, 3]
s_5 = ['s5', 10, 15, 500, 5, 7, 3]
s_6 = ['s6', 10, 15, 500, 8, 4, 3]
s_7 = ['s7', 20, 30, 500, 15, 7, 3]
s_8 = ['s8', 20, 30, 500, 20, 4, 3]
settings = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]  # all test settings

# 2.) enter the test number of the measurement that correspond to the test setting
sp_1 = ['TestNr_4.', 'TestNr_8.', 'TestNr_9.', 'TestNr_10.', 'TestNr_11.', 'TestNr_12.']
sp_2 = ['TestNr_13.', 'TestNr_14.', 'TestNr_15.', 'TestNr_16.', 'TestNr_17.']
sp_3 = ['TestNr_19.', 'TestNr_20.', 'TestNr_21.', 'TestNr_22.', 'TestNr_23.', 'TestNr_24.']
sp_4 = ['TestNr_31.', 'TestNr_26.', 'TestNr_27.', 'TestNr_32.', 'TestNr_29.', 'TestNr_30.']
sp_5 = ['TestNr_33.', 'TestNr_34.', 'TestNr_35.', 'TestNr_36.', 'TestNr_37.', 'TestNr_38.']
sp_6 = ['TestNr_39.', 'TestNr_40.', 'TestNr_41.', 'TestNr_42.', 'TestNr_43.', 'TestNr_44.']
sp_7 = ['TestNr_45.', 'TestNr_46.', 'TestNr_47.', 'TestNr_48.', 'TestNr_49.', 'TestNr_50.']
sp_8 = ['TestNr_51.', 'TestNr_52.', 'TestNr_53.', 'TestNr_54.', 'TestNr_55.', 'TestNr_56.']
specimen = [sp_1, sp_2, sp_3, sp_4, sp_5, sp_6, sp_7, sp_8]  # corresponding measurement numbers


# ------------------------------------------------------------------------------------------------------------------------
# ------------------------------------------------ FUNCTIONS -------------------------------------------------------------
```

```


def get_seconds(time_string):
    t = time_string.split('#')[1]
    if 'ms' in t:
        t = t.strip('ms')
        if 'm' in t and 's' in t:
            t = t.split('m')
            t1 = t[1].split('s')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t1[0]), milliseconds=int(t1[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 's' in t:
            t = t.split('s')
            time_ = timedelta(seconds=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        else:
            time_ = timedelta(milliseconds=int(t))
            time_ = time_.total_seconds()
    else:
        if 'm' in t and 's' in t:
            t = t.strip('s')
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.strip('m')
            time_ = timedelta(minutes=int(t))
            time_ = time_.total_seconds()
        elif 's':
            time_ = int(t.strip('s'))
    return time_


def read_measurement(filename_):
    # this function depends on structure of the measurement file
    time_ = []
    load_ = []
    abs_gap_ = []
    t_pyro_ = []
    t_thermo_ = []
    t_left_ = []
    with open(filename_, 'r') as f:
        header_ = f.readline()
        for line in f:
            d = line.strip().split(';')
            time_.append(get_seconds(d[0]))
            load_.append(float(d[1]))
            abs_gap_.append(float(d[2]))
            t_pyro_.append(float(d[4]))
            t_thermo_.append(float(d[5]))
            t_left_.append(float(d[6]))
    return header_, time_, load_, abs_gap_, t_pyro_, t_thermo_, t_left_


```

```python
def plot_layout(settings_, m):
    fig = plt.figure(figsize=(10, 10))
    sub1_ = fig.add_subplot(221)
    sub1_.set_title('Temperature during contact')
    sub1_.set_ylabel('Temperature °C')
    sub1_.set_xlabel('Time [s]')
    sub1_.set_ylim(0, settings_[m][3])

    sub2_ = fig.add_subplot(222)
    sub2_.set_title('Temperature during upsetting')
    sub2_.set_ylabel('Temperature °C')
    sub2_.set_xlabel('Time [s]')
    sub2_.set_ylim(0, settings_[m][3])

    sub3_ = fig.add_subplot(224)
    sub3_.set_title('Force during upsetting')
    sub3_.set_ylabel('Force [kN]')
    sub3_.set_xlabel('Upsetting height [mm]')

    title = (settings_[m][0] + ': specimen geometry: d0 = ' + str(settings_[m][1]) + ' [mm] h0= ' + str(settings_[m][2])
             + ' [mm], T= ' + str(settings_[m][3]) + ' [°C], Δh = ' + str(settings_[m][4]) +
             ' [mm], transfer time = ' + str(settings_[m][5]) + ' [s], rest time = ' + str(settings_[m][6]) + ' [s]')
    plt.suptitle(title)
    return sub1_, sub2_, sub3_


def start_contact(time_, t_):
    # at a specified point of time
    for ind in range(len(time_)):
        if time_[ind] > t_:
            return ind, time_[ind]


def start_upsetting(time_, load_):
    # determine the start of the upsetting process in the measurements
    # the start of the upsetting process is defined by the force exceeding a threshold value
    threshold = 0.1
    for ind in range(0, len(load_)):
        if load_[ind] > threshold:
            return ind, time_[ind]


def end_upsetting(time_, abs_gap_):
    # determine the end of the upsetting process in the measurements
    # the end of the upsetting process is reached with minimum relative gap between the dies
    minimum = min(abs_gap_)
    ind = abs_gap_.index(minimum)
    return ind, time_[ind]


def end_upsetting_max_force(time_, load_):  # todo optional method
    peak = max(load_)
    ind = load_.index(peak)
    return ind, time_[ind]


def evaluate_temperature(path, pyrometer_position, tolerance):
    # evaluate the temperature at a specified pyrometer position from .csv file from simulation
    z_coord, time_, temperature = np.loadtxt(path, delimiter=';', unpack=True)
    mask = (z_coord < pyrometer_position + tolerance) & (z_coord > pyrometer_position - tolerance)
    return time_[mask], temperature[mask]


def evaluate_force(path):
    # evaluate force from .csv file from simulation
    time_, u3, rf3 = np.loadtxt(path, delimiter=';', unpack=True)
    return -u3, -rf3 * 0.001  # unit of force [N] --> [kN]


def linear_fit(xfit, yfit):
    # linear fit to determine temperature at the end of the transport
    model = LinearModel()
    par_guess = model.guess(yfit, xfit)
    fitted = model.fit(yfit, par_guess, x=xfit)
    x_new = np.linspace(0, xfit[-1], 10)
    y_new = model.eval(params=fitted.params, x=x_new)
    return x_new, y_new

# --------------------------------------------- CALCULATIONS -----------------------------------------------------------


files = os.listdir(MEASUREMENT_FOLDER)
sim_files = s_files = os.listdir(SIMULATION_RESULTS_FOLDER)

plt.close('all')
# print('temperature after transport for each measurement')

nr = len(settings)
for i in range(nr):
    sub1, sub2, sub3 = plot_layout(settings, i)
    for j in range(len(specimen[i])):  # plot measurements
        for file in files:
            if specimen[i][j] in file:  # search for the test setting
                filepath = os.path.join(MEASUREMENT_FOLDER, file)
                header, time, load, abs_gap, t_pyro, t_thermo, t_left = read_measurement(filepath)

                for number, elem in enumerate(load):
                    load[number] = elem * 1.27

                filename = file.strip('.csv')

                # find start of contact to bottom die
                t1_i, t1 = start_contact(time, settings[i][5])

                # find start of the upsetting process
                t2_i, t2 = start_upsetting(time, load)

                # find end of the upsetting process
                t3_i, t3 = end_upsetting(time, abs_gap)

                # shift time to start at t=0 in subplot 1
                time_new = time
                for n, t in enumerate(time):
                    time_new[n] = time[n] - t1

                sub1.plot(time_new[t1_i:t2_i], t_pyro[t1_i:t2_i], label=filename)

                # fit to determine temperature at the end of the transport
                t_fit_i = int((t2_i + t1_i) / 2)  # define index between t2_i and t1_i for fit
                x, y = linear_fit(time_new[t_fit_i:t2_i], t_pyro[t_fit_i:t2_i])
                if showFit:
                    sub1.plot(x, y, 'k+', label=(filename + ' linear fit'))
                    print(settings[i][0] + ': ' + filename + ', t=' + str(settings[i][5]) + 's, T=' +
                          str(int(y[0])) + '°C')

                # shift in time again to start at t=0 in subplot 2
                time_new = time
                for n, t in enumerate(time):
                    time_new[n] = time[n] - (t2 - t1)

                sub2.plot(time_new[t2_i:t3_i], t_pyro[t2_i:t3_i], label=filename)

                # calculate gap
                gap = []
                for n, elem in enumerate(abs_gap):
                    gap.append(-(elem - abs_gap[t2_i]))

                sub3.plot(gap[t2_i:t3_i], load[t2_i:t3_i], label=filename)

    for sim_file in sim_files:  # plot simulation results
        if settings[i][0] in sim_file:
            s_filepath = os.path.join(SIMULATION_RESULTS_FOLDER, sim_file)
            name = 'simulation_' + settings[i][0]
            # todo: check keywords
            if 'sim3_' in sim_file:
                t, T = evaluate_temperature(s_filepath, hp, tol)
                sub1.plot(t, T, '.', label=name)
            elif 'sim4i_' in sim_file:
                if 'force' in sim_file:
                    t, rf = evaluate_force(s_filepath)
                    sub3.plot(t, rf, '.', label=name)
                    print(str(settings[i][0]), ': sim4i, maximum force = ', '{:.2f}'.format(np.max(rf)), ' kN')
                else:
                    t, T = evaluate_temperature(s_filepath, hp, tol)
                    sub2.plot(t, T, '.', label=name)
            elif 'sim4e_' in sim_file:
                if 'force' in sim_file:
                    t, rf = evaluate_force(s_filepath)
                    sub3.plot(t, rf, '.', label=name)
                    print(str(settings[i][0]), ': sim4e, maximum force = ', '{:.2f}'.format(np.max(rf)), ' kN')
                else:
                    t, T = evaluate_temperature(s_filepath, hp, tol)
                    sub2.plot(t, T, '.', label=name)

    sub1.legend(loc='upper center', bbox_to_anchor=(0.5, -0.5))
    if SAVE:
        name = 'comparison_' + settings[i][0]
        name = os.path.join(OUTPUT_FOLDER, name)
        plt.savefig(name, dpi=600)

plt.show()

```

```
# -------------------------------------------- SCRIPT INFORMATION ---------------------------------------------------
#
# name: compare_1502022
# function: compare experiment and simulation
# script includes the experiment information (section FILEPATH / EXPERIMENTS) for experiment 2

# -------------------------------------------- IMPORT -------------------------------------------------------------

import matplotlib.pyplot as plt
import numpy as np
import os
from datetime import timedelta
from lmfit.models import LinearModel

# -------------------------------------------- FILE PATHS ---------------------------------------------------------

MAIN_DIR = r'L:\090_Datenaustausch\cwaiguny\MA\project_15072022'
MEASUREMENT_FOLDER = os.path.join(MAIN_DIR, r'experiment\sensor')
SIMULATION_RESULTS_FOLDER = os.path.join(MAIN_DIR, r'sim\csv')
OUTPUT_FOLDER = os.path.join(MAIN_DIR, r'results')

SAVE = True  # if true, the figures are saved
showFit = False  # include linear fit in the plot

# -------------------------------------------- PARAMETER ----------------------------------------------------------

hp = 6.0  # pyrometer position [mm]
tol = 0.5  # tolerance for pyrometer position [mm]

# -------------------------------------------- EXPERIMENTS --------------------------------------------------------
# 1.) define parameter for each test setting: setting name/number, initial diameter d0 [mm], initial
height h0 [mm],
# furnace temperature [°C], upsetting height [mm], transport time [s], rest time on bottom die [s]

s_1 = ['s1', 10, 15, 300, 5, 4, 3]
s_2 = ['s2', 10, 15, 300, 5, 7, 3]
s_3 = ['s3', 10, 15, 400, 5, 4, 3]
s_4 = ['s4', 10, 15, 400, 5, 7, 3]
s_5 = ['s5', 10, 15, 500, 5, 4, 3]
s_6 = ['s6', 10, 15, 500, 5, 7, 3]
settings = [s_1, s_2, s_3, s_4, s_5, s_6]  # all test settings

# 2.) enter the test number of the measurement that correspond to the test setting
sp_1 = ['TestNr_12.', 'TestNr_13.', 'TestNr_14.', 'TestNr_15.']
sp_2 = ['TestNr_16.', 'TestNr_17.', 'TestNr_18.', 'TestNr_19.']
sp_3 = ['TestNr_20.', 'TestNr_21.', 'TestNr_22.', 'TestNr_23.']
sp_4 = ['TestNr_24.', 'TestNr_25.', 'TestNr_26.', 'TestNr_27.']
sp_5 = ['TestNr_28.', 'TestNr_29.', 'TestNr_30.', 'TestNr_31.']
sp_6 = ['TestNr_32.', 'TestNr_33.', 'TestNr_34.', 'TestNr_35.']
specimen = [sp_1, sp_2, sp_3, sp_4, sp_5, sp_6]


# ------------------------------------------------------------------------------------------------------------------
# -------------------------------------------- FUNCTIONS ----------------------------------------------------------


def get_seconds(time_string):
    t = time_string.split('#')[1]
    if 'ms' in t:
        t = t.strip('ms')
        if 'm' in t and 's' in t:
            t = t.split('m')
            t1 = t[1].split('s')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t1[0]), milliseconds=int(t1[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 's' in t:
            t = t.split('s')
            time_ = timedelta(seconds=int(t[0]), milliseconds=int(t[1]))
            time_ = time_.total_seconds()
        else:
            time_ = timedelta(milliseconds=int(t))
            time_ = time_.total_seconds()
    else:
        if 'm' in t and 's' in t:
            t = t.strip('s')
            t = t.split('m')
            time_ = timedelta(minutes=int(t[0]), seconds=int(t[1]))
            time_ = time_.total_seconds()
        elif 'm' in t:
            t = t.strip('m')
            time_ = timedelta(minutes=int(t))
            time_ = time_.total_seconds()
        elif 's':
            time_ = int(t.strip('s'))
    return time_


def read_measurement(filename_):
    # this function depends on structure of the measurement file
    time_ = []
    load_ = []
    abs_gap_ = []
    t_pyro_ = []
    t_thermo_ = []
    t_left_ = []
    with open(filename_, 'r') as f:
        header_ = f.readline()
        for line in f:
            d = line.strip().split(';')
            time_.append(get_seconds(d[0]))
            load_.append(float(d[1]))
            abs_gap_.append(float(d[2]))
            t_pyro_.append(float(d[4]))
            t_thermo_.append(float(d[6]))
            t_left_.append(float(d[7]))
    return header_, time_, load_, abs_gap_, t_pyro_, t_thermo_, t_left_


def plot_layout(settings_, m):
    fig = plt.figure(figsize=(10, 10))
    sub1_ = fig.add_subplot(221)
    sub1_.set_title('Temperature during contact to bottom die')
    sub1_.set_ylabel('Temperature °C')
    sub1_.set_xlabel('Time [s]')
    sub1_.set_ylim(0, settings_[m][3])

    sub2_ = fig.add_subplot(222)
    sub2_.set_title('Temperature during upsetting')
    sub2_.set_ylabel('Temperature °C')
    sub2_.set_xlabel('Time [s]')
    sub2_.set_ylim(0, settings_[m][3])

    sub3_ = fig.add_subplot(224)
    sub3_.set_title('Force during upsetting')
    sub3_.set_ylabel('Force [kN]')
    sub3_.set_xlabel('Upsetting height [mm]')

    title = (settings_[m][0] + ': specimen geometry: d0 = ' + str(settings_[m][1]) + ' [mm] h0= ' + str(
settings_[m][2])
             + ' [mm], T= ' + str(settings_[m][3]) + ' [°C], Δh = ' + str(settings_[m][4]) +
             ' [mm], transfer time = ' + str(settings_[m][5]) + ' [s], rest time = ' + str(settings_[m][6]) + ' [s]')
    plt.suptitle(title)
    return sub1_, sub2_, sub3_


def start_contact(time_, t_):
    # at a specified point of time
    for ind in range(len(time_)):
        if time_[ind] > t_:
            return ind, time_[ind]


def start_upsetting(time_, load_):
    # determine the start of the upsetting process in the measurements
    # the start of the upsetting process is defined by the force exceeding a threshold value
    threshold = 0.1
    for ind in range(0, len(load_)):
        if load_[ind] > threshold:
            return ind, time_[ind]


def end_upsetting(time_, abs_gap_):
    # determine the end of the upsetting process in the measurements
    # the end of the upsetting process is reached with minimum relative gap between the dies
    minimum = min(abs_gap_)
    ind = abs_gap_.index(minimum)
    return ind, time_[ind]


def end_upsetting_max_force(time_, load_):  # todo optional method
    peak = max(load_)
    ind = load_.index(peak)
    return ind, time_[ind]


def evaluate_temperature(path, pyrometer_position, tolerance):
    # evaluate the temperature at a specified pyrometer position from .csv file from simulation
    z_coord, time_, temperature = np.loadtxt(path, delimiter=';', unpack=True)
    mask = (z_coord < pyrometer_position + tolerance) & (z_coord > pyrometer_position - tolerance)
    return time_[mask], temperature[mask]
```

```


def evaluate_force(path):
    # evaluate force from .csv file from simulation
    time_, u3, rf3 = np.loadtxt(path, delimiter=';', unpack=True)
    return -u3, -rf3 * 0.001  # unit of force [N] --> [kN]


def linear_fit(xfit, yfit):
    # linear fit to determine temperature at the end of the transport
    model = LinearModel()
    par_guess = model.guess(yfit, xfit)
    fitted = model.fit(yfit, par_guess, x=xfit)
    x_new = np.linspace(0, xfit[-1], 10)
    y_new = model.eval(params=fitted.params, x=x_new)
    return x_new, y_new


# ------------------------------------------------ CALCULATIONS ------------------------------------------------------------

files = os.listdir(MEASUREMENT_FOLDER)
sim_files = s_files = os.listdir(SIMULATION_RESULTS_FOLDER)

plt.close('all')
# print('temperature after transport for each measurement')

nr = len(settings)
for i in range(nr):
    sub1, sub2, sub3 = plot_layout(settings, i)
    for j in range(len(specimen[i])):  # plot measurements
        for file in files:
            if specimen[i][j] in file:  # search for the test setting
                filepath = os.path.join(MEASUREMENT_FOLDER, file)
                header, time, load, abs_gap, t_pyro, t_thermo, t_left = read_measurement(filepath)

                filename = file.strip('.csv')

                # find start of contact to bottom die
                t1_i, t1 = start_contact(time, settings[i][5])

                # find start of the upsetting process
                t2_i, t2 = start_upsetting(time, load)

                # find end of the upsetting process
                t3_i, t3 = end_upsetting(time, abs_gap)

                # shift time to start at t=0 in subplot 1
                time_new = time
                for n, t in enumerate(time):
                    time_new[n] = time[n] - t1

                sub1.plot(time_new[t1_i:t2_i], t_pyro[t1_i:t2_i], label=filename)

                # fit to determine temperature at the end of the transport
                t_fit_i = int((t2_i + t1_i) / 2)  # define index between t2_i and t1_i for fit
                x, y = linear_fit(time_new[t_fit_i:t2_i], t_pyro[t_fit_i:t2_i])
                if showFit:
                    sub1.plot(x, y, 'k+', label=(filename + ' linear fit'))
                    print(settings[i][0] + ': ' + filename + ', t= ' + str(settings[i][5]) + 's, T= ' +
                          str(int(y[0])) + '°C')

                # shift in time again to start at t=0 in subplot 2
                time_new = time
                for n, t in enumerate(time):
                    time_new[n] = time[n] - (t2 - t1)

                sub2.plot(time_new[t2_i:t3_i], t_pyro[t2_i:t3_i], label=filename)

                # calculate gap
                gap = []
                for n, elem in enumerate(abs_gap):
                    gap.append(-(elem - abs_gap[t2_i]))

                sub3.plot(gap[t2_i:t3_i], load[t2_i:t3_i], label=filename)

    for sim_file in sim_files:  # plot simulation results
        if settings[i][0] in sim_file:
            s_filepath = os.path.join(SIMULATION_RESULTS_FOLDER, sim_file)
            name = 'simulation_' + settings[i][0]
            # todo: check keywords
            if 'sim3_' in sim_file:
                t, T = evaluate_temperature(s_filepath, hp, tol)
                sub1.plot(t, T, '.', label=name)
            elif 'sim4i_' in sim_file:
                if 'force' in sim_file:
                    t, rf = evaluate_force(s_filepath)
                    sub3.plot(t, rf, '.', label=name)
                    print(str(settings[i][0]), ': sim4i, maximum force = ', '{:.2f}'.format(np.max(rf)), ' kN')
                else:
                    t, T = evaluate_temperature(s_filepath, hp, tol)
                    sub2.plot(t, T, '.', label=name)
            elif 'sim4e_' in sim_file:
                if 'force' in sim_file:
                    t, rf = evaluate_force(s_filepath)
                    sub3.plot(t, rf, '.', label=name)
                    print(str(settings[i][0]), ': sim4e, maximum force = ', '{:.2f}'.format(np.max(rf)), ' kN')
                else:
                    t, T = evaluate_temperature(s_filepath, hp, tol)
                    sub2.plot(t, T, '.', label=name)

    sub1.legend(loc='upper center', bbox_to_anchor=(0.5, -0.5))
    if SAVE:
        name = 'comparison_' + settings[i][0]
        name = os.path.join(OUTPUT_FOLDER, name)
        plt.savefig(name, dpi=600)

plt.show()

```

## Appendix D: simulation_1.py

```
# ---------------------------------------------- SCRIPT INFORMATION -----------------------------------------------------
#
# name: simulation_1:
# function: heating of the specimen in the furnace
#
# ---------------------------------------------- IMPORT ----------------------------------------------------------------

from modules import abaqus_functions
from modules import material_data

# ---------------------------------------------- INPUT PARAMETER --------------------------------------------------------
# unit system: SI-mm

# h0                                # initial height of the specimen (z-dimension)
# d0                                # initial diameter of specimen

# ambient_temperature                    # ambient temperature
# furnace_temperature                   # initial temperature of the preheated furnace

# heating_time                      # process time for heating

# amplitude_temperature_data                # time-temperature amplitude of the furnace

# thermal_conductance                   # thermal conductance for contact area between specimen and
furnace
# emissivity                        # emissivity of the specimen
# emissivity_f                      # emissivity of the furnace lining
# heat_transfer_coeff                   # heat transfer coefficient  for convective heat transfer

# seed_size                         # global seed size applied to the specimen
# seed_size_furnace                     # global seed size applied to the furnace

# result_path                       # absolute file path of the folder in which the results are saved

# name                              # name of the Abaqus model, step and job

# ----------------------------------------------SIMULATION 1 ------------------------------------------------------------


def heating(h0, d0, ambient_temperature, furnace_temperature, heating_time, emissivity, emissivity_f,
         heat_transfer_coeff, amplitude_temperature_data, thermal_conductance, seed_size,
seed_size_furnace,
         result_path, name):

   model_name = name
   model = abaqus_functions.model_settings(model_name)

   # ---------------------------------------------- PART MODULE --------------------------------------------------------
   # create parts
   specimen = abaqus_functions.create_cylinder(model, 'specimen', d0, h0)
   x_dim_inside, y_dim_inside, z_dim_inside, thickness_walls, thickness_bottom_wall = \
      abaqus_functions.get_dimensions_furnace()
   furnace = abaqus_functions.create_furnace(model, 'furnace', x_dim_inside, y_dim_inside,
z_dim_inside,
                                 thickness_walls, thickness_bottom_wall)

   # create sets
   set_specimen = abaqus_functions.create_set_all(specimen, 'set_specimen')
```

```
set_furnace = abaqus_functions.create_set_all(furnace, 'set_furnace')

# ------------------------------------------- PROPERTY MODULE ------------------------------------------------
# material definition for the specimen
aluminum = abaqus_functions.create_material(model, 'aluminum')
data_1 = material_data.material_aluminum()
abaqus_functions.define_density(aluminum, data_1[0], data_1[1])
abaqus_functions.define_conductivity(aluminum, data_1[2], data_1[3])
abaqus_functions.define_specific_heat(aluminum, data_1[4], data_1[5])

# material definition for furnace
refractory = abaqus_functions.create_material(model, 'refractory')
data_2 = material_data.material_refractory()
abaqus_functions.define_density(refractory, data_2[0], data_2[1])
abaqus_functions.define_conductivity(refractory, data_2[2], data_2[3])
abaqus_functions.define_specific_heat(refractory, data_2[4], data_2[5])

# create and assign sections
abaqus_functions.create_section(model, 'section_aluminum', 'aluminum')
abaqus_functions.assign_section(specimen, set_specimen, 'section_aluminum')
abaqus_functions.create_section(model, 'section_refractory', 'refractory')
abaqus_functions.assign_section(furnace, set_furnace, 'section_refractory')

# ------------------------------------------- ASSEMBLY MODULE ------------------------------------------------
# create assembly
inst_specimen, inst_furnace = abaqus_functions.create_assembly_furnace(model, specimen, furnace,
                                                      thickness_bottom_wall)

# ------------------------------------------- STEP MODULE ----------------------------------------------------
# create step
abaqus_functions.create_heat_transfer_step(model, name, 'Initial', heating_time)

# create field output
abaqus_functions.delete_f_output1(model)  # delete automatic output
f_output_all = 'NT', 'HFL'  # field output variables for whole model
time_interval = 60  # time interval for the field output
abaqus_functions.create_field_output(model, 'f_output_whole_model', name, f_output_all, time_interval)

# ------------------------------------------- INTERACTION MODULE ---------------------------------------------
# create amplitude with temperature data
abaqus_functions.create_amplitude(model, 'amplitude_temperature', amplitude_temperature_data)

# create surfaces
surfaces_specimen = abaqus_functions.create_surfaces(model, inst_specimen, 'surfaces_specimen',
                                         (d0 / 2, 0, h0 / 2), (0, 0, h0))
bottom_surf_specimen = abaqus_functions.create_surfaces(model, inst_specimen, 'bottom_surface_specimen',
                                           (0, 0, 0))
bottom_surf_furnace = abaqus_functions.create_surfaces(model, inst_furnace, 'bottom_surf_furnace', (0, 0, 0))
inside_surfaces = abaqus_functions.create_surfaces(model, inst_furnace, 'inside_surfaces', (0, 0, 0),
                                        (x_dim_inside / 2, 0, z_dim_inside / 2),
                                        (-x_dim_inside / 2, 0, z_dim_inside / 2),
                                        (0, y_dim_inside / 2, z_dim_inside / 2),
                                        (0, -y_dim_inside / 2, z_dim_inside / 2), (0, 0, z_dim_inside))
```

```

    # radiation
    abaqus_functions.create_radiation_to_var_ambient(model, 'radiation_specimen', surfaces_specimen,
                                                     name, 'amplitude_temperature', emissivity)
    abaqus_functions.create_radiation_to_var_ambient(model, 'radiation_furnace', inside_surfaces,
                                                     name, 'amplitude_temperature', emissivity_f)

    # convection
    abaqus_functions.create_convection_var(model, 'convection_specimen', surfaces_specimen, name,
                                           'amplitude_temperature', heat_transfer_coeff)

    # contact
    abaqus_functions.create_contact_property_thermal(model, 'contact_property', thermal_conductance)
    abaqus_functions.create_contact_interaction(model, 'contact_interaction', bottom_surf_furnace,
                                                bottom_surf_specimen, 'contact_property')

    # ------------------------------------------- LOAD MODULE ------------------------------------------------------
    # define the initial temperatures
    abaqus_functions.create_predefined_field(model, 'init_temp_specimen', inst_specimen, 'set_specimen',
                                             ambient_temperature)
    abaqus_functions.create_predefined_field(model, 'init_temp_furnace', inst_furnace, 'set_furnace',
                                             furnace_temperature)

    # create sets
    set_5_walls_furnace = abaqus_functions.create_set_faces(model, 'set_5_walls_furnace', inst_furnace,
                                                            (x_dim_inside / 2, 0, z_dim_inside / 2),
                                                            (-x_dim_inside / 2, 0, z_dim_inside / 2),
                                                            (0, y_dim_inside / 2, z_dim_inside / 2),
                                                            (0, -y_dim_inside / 2, z_dim_inside / 2),
                                                            (0, 0, z_dim_inside))

    # create boundary conditions
    abaqus_functions.create_boundary_temperature(model, 'temperature_boundary', set_5_walls_furnace,
                                                 name, 'amplitude_temperature')

    # ------------------------------------------- MESH MODULE ------------------------------------------------------
    # create mesh
    abaqus_functions.create_partitions(specimen, h0)
    abaqus_functions.mesh_control_cylinder(specimen)
    abaqus_functions.create_mesh_1(specimen, seed_size)

    abaqus_functions.create_partitions_furnace(furnace, x_dim_inside, y_dim_inside, z_dim_inside,
                                               thickness_bottom_wall)
    abaqus_functions.create_mesh_1(furnace, seed_size_furnace)

    # create node set and output for node set
    abaqus_functions.create_node_set(specimen, 'set_nodes', seed_size, (d0 / 2, 0, 0), (d0 / 2, 0, h0))
    h_output = 'COORD', 'NT'
    time_interval = 60  # time interval for the history output
    abaqus_functions.create_history_output(model, inst_specimen, 'h_output', 'set_nodes', name,
                                           h_output, time_interval)

    # ------------------------------------------- JOB MODULE ------------------------------------------------------
    # create job and run simulation
    abaqus_functions.create_job_heat_transfer(model_name, name, 6)
    # abaqus_functions.write_input(name, result_path)
    abaqus_functions.submit_job(name, result_path)
    abaqus_functions.wait_for_job(name)

# ------------------------------------------------------------------------------------------------------------------------

```

## Appendix E: simulation_2p.py

```
# ------------------------------------------- SCRIPT INFORMATION --------------------------------------------------

# name: simulation_2p
# function: transport of the specimen from the furnace to the hydraulic press
# solving the differential equation for a 0-dimensional heat transfer problem

# ------------------------------------------- IMPORT ------------------------------------------------------------

import numpy as np
from modules import material_data

# ------------------------------------------- INPUT PARAMETER -----------------------------------------------------
# unit system: SI-mm

# h0                        # initial height of the specimen (z-dimension)
# d0                        # initial diameter of specimen

# transport_time                # time time for the transport

# Ta                        # ambient temperature
# T_init                     # initial temperature of the specimen

# epsilon                     # emissivity of the specimen
# h                        # heat transfer coefficient for convective heat transfer

# ------------------------------------------- SIMULATION 2 P ------------------------------------------------------


def transport(d0, h0, Ta, T_init, t, epsilon, h):
    dt = 0.1  # time delta

    # ------------------------------------------- PROPERTIES ------------------------------------------------------

    sigma = 5.67E-11  # Boltzmann constant
    k = material_data.conductivity_function(T_init)  # conductivity for initial temperature
    c = material_data.specific_heat_function(T_init)  # specific heat for initial temperature
    data_1 = material_data.material_aluminum()
    rho = data_1[0]

    # ------------------------------------------- CALCULATIONS ----------------------------------------------------

    V = d0 ** 2 * np.pi / 4 * h0             # specimen volume
    As = d0 * np.pi * h0 + 2 * d0 ** 2 * np.pi / 4  # specimen surface
    Lc = V / As                       # characteristic length

    Bi = h * Lc / k                    # Biot number
    if Bi > 0.1:                     # lumped mass approximation valid for Bi < 0.1
        print('Biot number is greater than 0.1 - approximation not valid')

    # solve energy balance equation
    nr_iterations = int(t / dt)
    T = np.zeros(nr_iterations)
    time = np.zeros(nr_iterations)
    T[0] = T_init

    A_cont = 20 + (d0/10 - 1) * 5             # fit for contact area
    q = 2.5 * T_init * (T_init/Ta) + (d0 / 10 - 1) * ((T_init**3)/890)  # fit for heat flux

    for i in range(0, nr_iterations - 1, 1):
        # considering radiation, convection and a correction term for the heat conduction to the gripper
        if i*dt < 2:
            T[i + 1] = T[i] + dt * (1 / (rho * V * c)) * (
                    h * As * (Ta - T[i]) +
                    epsilon * sigma * As * (Ta ** 4 - T[i] ** 4) -
                    q * A_cont)
            q = q * 0.95
        # considering radiation and convection
        elif i*dt >= 2:
            T[i + 1] = T[i] + dt * (1 / (rho * V * c)) * (
                    h * As * (Ta - T[i]) +
                    epsilon * sigma * As * (Ta ** 4 - T[i] ** 4))

        time[i + 1] = time[i] + dt

    T_end = T[-1]

    return T_end

# -------------------------------------------------------------------------------------------------------------------


```

## Appendix F: simulation_3p.py

```
# ---------------------------------------------- SCRIPT INFORMATION -------------------------------------------------------

# name: simulation_3p
# function: heat transfer while specimen rests on the bottom die

# ---------------------------------------------- IMPORT ---------------------------------------------------------------

from modules import abaqus_functions
from modules import material_data

# ---------------------------------------------- INPUT PARAMETER ------------------------------------------------------
# unit system: SI-mm

# h0                                # initial height of the specimen (z-dimension)
# d0                                # initial diameter of specimen

# time                              # process time

# ambient_temperature                    # ambient temperature
# init_temperature                   # temperature of the specimen after heating

# thermal_conductance                   # thermal conductance for contact area between specimen and die
# emissivity                        # emissivity
# heat_transfer_coeff                   # heat transfer coefficient  for convective heat transfer

# seed_size                         # global seed size applied to the specimen
# seed_size_die                       # global seed size applied to the die

# result_path                       # absolute file path of the folder in which the results are saved

# name                              # name of the Abaqus model, step and job

# ---------------------------------------------------------------------------------------------------------------------


def rest(h0, d0, time, emissivity, ambient_temperature, init_temperature, thermal_conductance, heat_transfer_coeff,
        seed_size, seed_size_die, result_path, name):

    model_name = name
    model = abaqus_functions.model_settings(model_name)

    # ---------------------------------------------- PART MODULE ----------------------------------------------------
    # create parts
    specimen = abaqus_functions.create_cylinder(model, 'specimen', d0, h0)
    bottom_die = abaqus_functions.create_die(model, 'bottom_die', d0, d0)

    # create sets
    set_specimen = abaqus_functions.create_set_all(specimen, 'set_specimen')
    set_bottom_die = abaqus_functions.create_set_all(bottom_die, 'set_bottom_die')

    # ---------------------------------------------- PROPERTY MODULE ------------------------------------------------
    # material definition for the specimen
    aluminum = abaqus_functions.create_material(model, 'aluminum')
    data_1 = material_data.material_aluminum()
    abaqus_functions.define_density(aluminum, data_1[0], data_1[1])
    abaqus_functions.define_conductivity(aluminum, data_1[2], data_1[3])
    abaqus_functions.define_specific_heat(aluminum, data_1[4], data_1[5])

    # material definition for the die
    steel = abaqus_functions.create_material(model, 'steel')
    data_2 = material_data.material_steel()
    abaqus_functions.define_density(steel, data_2[0], data_2[1])
    abaqus_functions.define_conductivity(steel, data_2[2], data_2[3])
    abaqus_functions.define_specific_heat(steel, data_2[4], data_2[5])

    # create and assign sections
    abaqus_functions.create_section(model, 'section_aluminum', 'aluminum')
    abaqus_functions.assign_section(specimen, set_specimen, 'section_aluminum')
    abaqus_functions.create_section(model, 'section_steel', 'steel')
    abaqus_functions.assign_section(bottom_die, set_bottom_die, 'section_steel')

    # -------------------------------------------- ASSEMBLY MODULE ----------------------------------------------

    inst_specimen, inst_bottom_die = abaqus_functions.create_assembly_2parts(model, specimen, bottom_die,
                                                                             'instance_specimen',
                                                                             'instance_bottom_die', 20)

    # -------------------------------------------- STEP MODULE ----------------------------------------------
    # create step
    abaqus_functions.create_heat_transfer_step(model, name, 'Initial', time)

    # create field output
    f_output_all = 'NT', 'HFL', 'COORD'  # field output variables for whole model
    time_interval = 1  # time interval for the field output
    abaqus_functions.create_field_output(model, 'f_output_whole_model', name, f_output_all, time_interval)

    # -------------------------------------------- INTERACTION MODULE ----------------------------------------------

    # create surfaces
    surfaces_specimen = abaqus_functions.create_surfaces(model, inst_specimen, 'surfaces_specimen',
                                                         (d0 / 2, 0, h0 / 2), (0, 0, h0))
    contact_surf_specimen = abaqus_functions.create_surfaces(model, inst_specimen, 'contact_surface_specimen',
                                                             (0, 0, 0))
    contact_surf_die = abaqus_functions.create_surfaces(model, inst_bottom_die, 'contact_surface_die', (0, 0, 0))

    # contact
    abaqus_functions.create_contact_property_thermal(model, 'contact_property', thermal_conductance)
    abaqus_functions.create_contact_interaction(model, 'contact_interaction', contact_surf_die, contact_surf_specimen,
                                                'contact_property')

    # radiation
    abaqus_functions.create_radiation_to_ambient(model, 'radiation_specimen', surfaces_specimen,
                                                 name, ambient_temperature, emissivity)

    # convection
    abaqus_functions.create_convection(model, 'convection_specimen', surfaces_specimen, name,
                                       heat_transfer_coeff, ambient_temperature)

    # -------------------------------------------- LOAD MODULE -------------------------------------------------------
    # initial temperatures
    abaqus_functions.create_predefined_field(model, 'init_temp_bottom_die', inst_bottom_die, '
set_bottom_die',
                                ambient_temperature)
    abaqus_functions.create_predefined_field(model, 'init_temp_specimen', inst_specimen, '
set_specimen',
                                init_temperature)

    # -------------------------------------------- MESH MODULE -------------------------------------------------------
    # mesh
    abaqus_functions.create_partitions(specimen, h0)
    abaqus_functions.mesh_control_cylinder(specimen)

    abaqus_functions.create_mesh_1(specimen, seed_size)
    abaqus_functions.create_mesh_1(bottom_die, seed_size_die)

    # node set and history output
    h_output_nodes = 'COOR3', 'NT'
    time_interval = 0.5  # time interval for the history output
    abaqus_functions.create_node_set(specimen, 'set_nodes', seed_size, (d0 / 2, 0, 0), (d0 / 2, 0, h0))
    abaqus_functions.create_history_output(model, inst_specimen, 'h_output_nodes', 'set_nodes',
name, h_output_nodes,
                               time_interval)

    # -------------------------------------------- JOB MODULE -------------------------------------------------------
    # create job and write input file
    abaqus_functions.create_job_heat_transfer(model_name, name, 2)
    # abaqus_functions.write_input(name, result_path)
    abaqus_functions.submit_job(name, result_path)
    abaqus_functions.wait_for_job(name)

# ------------------------------------------------------------------------------------------------------------------

```

## Appendix G: simulation_4i.py

```python
# -------------------------------------------- SCRIPT INFORMATION ----------------------------------------------------

# name: simulation_4i
# function: upsetting of the specimen (implicit solver)
#
# -------------------------------------------- IMPORT --------------------------------------------------------------

from modules import abaqus_functions
from modules import material_data

# -------------------------------------------- INPUT PARAMETER ---------------------------------------------------
# unit system: SI-mm

# h0                                # initial height of the specimen (z-dimension)
# d0                                # initial diameter of specimen

# ambient_temperature                       # ambient temperature

# time                              # process time

# friction_coefficient                  # friction coefficient between specimen and die
# thermal_conductance                       # thermal conductance for contact area between specimen and die

# odb_abs_path                      # file path of odb from previous simulation
# amplitude_displacement_data               # time-displacement data for the movement of the top die

# seed_size                         # global seed size applied to the specimen
# seed_size_die                       # global seed size applied to the dies

# result_path                        # absolute file path for the results
# odb_abs_path                         # absolute file path of the odb

# name                              # name of the Abaqus model, step and job

# ---------------------------------------------------------------------------------------------------------------


def upsetting(h0, d0, ambient_temperature, time, friction_coefficient, thermal_conductance, odb_abs_path,
          amplitude_displacement_data, seed_size, seed_size_die, result_path, name):

   model_name = name
   model = abaqus_functions.model_settings(model_name)

   # -------------------------------------------- PART MODULE ---------------------------------------------------
   # create parts
   specimen = abaqus_functions.create_cylinder(model, 'specimen', d0, h0)
   top_die = abaqus_functions.create_die(model, 'top_die', d0, d0)
   bottom_die = abaqus_functions.create_die(model, 'bottom_die', d0, d0)

   # create sets
   set_specimen = abaqus_functions.create_set_all(specimen, 'set_specimen')
   set_top_die = abaqus_functions.create_set_all(top_die, 'set_top_die')
   set_bottom_die = abaqus_functions.create_set_all(bottom_die, 'set_bottom_die')

   # -------------------------------------------- PROPERTY MODULE ---------------------------------------------------
   # material definition for the specimen
```

```python
aluminum = abaqus_functions.create_material(model, 'aluminum')
data_1 = material_data.material_aluminum()
abaqus_functions.define_density(aluminum, data_1[0], data_1[1])
abaqus_functions.define_conductivity(aluminum, data_1[2], data_1[3])
abaqus_functions.define_specific_heat(aluminum, data_1[4], data_1[5])
abaqus_functions.define_elasticity(aluminum, data_1[6], data_1[7], data_1[8])
abaqus_functions.define_plasticity(aluminum, data_1[9], data_1[10])
abaqus_functions.define_inelastic_heat_fraction(aluminum, data_1[12])

# material definition for the dies
steel = abaqus_functions.create_material(model, 'steel')
data_2 = material_data.material_steel()
abaqus_functions.define_density(steel, data_2[0], data_2[1])
abaqus_functions.define_conductivity(steel, data_2[2], data_2[3])
abaqus_functions.define_specific_heat(steel, data_2[4], data_2[5])
abaqus_functions.define_elasticity(steel, data_2[6], data_2[7], data_2[8])

# create and assign sections
abaqus_functions.create_section(model, 'section_aluminum', 'aluminum')
abaqus_functions.assign_section(specimen, set_specimen, 'section_aluminum')
abaqus_functions.create_section(model, 'section_steel', 'steel')
abaqus_functions.assign_section(top_die, set_top_die, 'section_steel')
abaqus_functions.assign_section(bottom_die, set_bottom_die, 'section_steel')

# ------------------------------------------- ASSEMBLY MODULE ------------------------------------------------
# create assembly
inst_specimen, inst_bottom_die, inst_top_die = abaqus_functions.create_assembly_press(model, specimen, bottom_die,
                                                                      top_die, h0)
# create reference point
set_rp = abaqus_functions.create_ref_point(model, 'set_rp', h0)

# ------------------------------------------- STEP MODULE ---------------------------------------------------
# create step
abaqus_functions.coupled_tep_displ_step(model, name, time)

# create field and history output
abaqus_functions.delete_automatic_output(model)
# output variables
f_output_all = 'S', 'U', 'NT', 'PE', 'PEEQ', 'COORD', 'CSTRESS', 'CFORCE', 'HFL'
h_output = 'U3', 'RF3'  # history output for the reference node
num_interval = 10  # number of intervals for the field output
abaqus_functions.create_field_output_2(model, 'f_output_whole_model', name, f_output_all, num_interval)
num_interval_rp = 100  # number of intervals for the history output of the reference point
abaqus_functions.create_history_output_rp(model, 'h_output', 'set_rp', name, h_output, num_interval_rp)

# ------------------------------------------- INTERACTION MODULE ---------------------------------------------
# create general contact between specimen and dies
abaqus_functions.create_contact_property(model, 'contact_property', friction_coefficient, thermal_conductance)
abaqus_functions.create_general_contact(model, 'general_contact', 'contact_property')

# create kinematic coupling between reference point and top die
top_die_surface = abaqus_functions.create_surfaces(model, inst_top_die, 'top_die_surface', (0, 0, h0 + 0.1))
abaqus_functions.create_kin_coupling(model, 'kinematic_coupling', set_rp, top_die_surface)
```

```

    # ---------------------------------------------- LOAD MODULE ----------------------------------------------------------
    # define the initial temperatures
    abaqus_functions.create_predefined_field(model, 'init_temp_top_die', inst_top_die, 'set_top_die',
                                     ambient_temperature)
    abaqus_functions.create_predefined_field_from_output(model, 'init_temp_specimen_bottom_die'
, inst_specimen,
                                              'set_specimen', odb_abs_path)

    # create amplitude with displacement data
    abaqus_functions.create_amplitude(model, 'amplitude_displacement',
amplitude_displacement_data)

    # create boundaries: fix bottom die and apply displacement to top die
    abaqus_functions.create_boundary_fixed(model, 'boundary_fixed', inst_bottom_die, '
set_bottom_die')
    abaqus_functions.create_boundary_displacement(model, 'boundary_displacement', '
amplitude_displacement', set_rp,
                                       name)

    # ---------------------------------------------- MESH MODULE ----------------------------------------------------------
    # create mesh
    abaqus_functions.create_partitions(specimen, h0)
    abaqus_functions.mesh_control_cylinder(specimen)
    abaqus_functions.create_mesh_2(specimen, seed_size)

    abaqus_functions.create_mesh_2(top_die, seed_size_die)
    abaqus_functions.create_mesh_2(bottom_die, seed_size_die)

    # create node set and output for node set
    abaqus_functions.create_node_set(specimen, 'set_nodes', seed_size, (d0 / 2, 0, 0), (d0 / 2, 0, h0))
    h_output_nodes = 'COOR3', 'NT'
    time_interval = 0.05  # time interval for the history output
    abaqus_functions.create_history_output(model, inst_specimen, 'h_output_nodes', 'set_nodes',
name, h_output_nodes,
                                  time_interval)

    # ---------------------------------------------- JOB MODULE ----------------------------------------------------------
    # create job run simulation
    abaqus_functions.create_job_upsetting(model_name, name, 6)
    # abaqus_functions.write_input(name, result_path)
    abaqus_functions.submit_job(name, result_path)
    abaqus_functions.wait_for_job(name)

# -----------------------------------------------------------------------------------------------------------------------

```

## Appendix H: simulation_4e.py

```python
# ------------------------------------------ SCRIPT INFORMATION ------------------------------------------------

# name: simulation_4e
# function: upsetting of the specimen (explicit solver)
#
# ------------------------------------------ IMPORT -------------------------------------------------------

from modules import abaqus_functions
from modules import material_data

# ------------------------------------------ INPUT PARAMETER ------------------------------------------------
# unit system: SI-mm

# h0                                # initial height of the specimen (z-dimension)
# d0                                # initial diameter of specimen

# ambient_temperature               # ambient temperature

# time                              # process time

# friction_coefficient              # friction coefficient between specimen and die
# thermal_conductance               # thermal conductance for contact area between specimen and die

# odb_abs_path                      # file path of odb from previous simulation
# amplitude_displacement_data       # time-displacement data for the movement of the top die

# seed_size                         # global seed size applied to the specimen
# seed_size_die                     # global seed size applied to the dies

# result_path                       # absolute file path for the results
# odb_abs_path                      # absolute file path of the odb

# name                              # name of the Abaqus model, step and job

# ---------------------------------------------------------------------------------------------------------


def upsetting(h0, d0, ambient_temperature, time, friction_coefficient, thermal_conductance, odb_abs_path,
          amplitude_displacement_data, seed_size, seed_size_die, result_path, name):
   model_name = name
   model = abaqus_functions.model_settings(model_name)

   # ------------------------------------------ PART MODULE ------------------------------------------------
   # create parts
   specimen = abaqus_functions.create_cylinder(model, 'specimen', d0, h0)
   top_die = abaqus_functions.create_die(model, 'top_die', d0, d0)
   bottom_die = abaqus_functions.create_die(model, 'bottom_die', d0, d0)

   # create sets
   set_specimen = abaqus_functions.create_set_all(specimen, 'set_specimen')
   set_top_die = abaqus_functions.create_set_all(top_die, 'set_top_die')
   set_bottom_die = abaqus_functions.create_set_all(bottom_die, 'set_bottom_die')

   # ------------------------------------------ PROPERTY MODULE ------------------------------------------------
   # material definition for the specimen
   aluminum = abaqus_functions.create_material(model, 'aluminum')
```

```
data_1 = material_data.material_aluminum()
abaqus_functions.define_density(aluminum, data_1[0], data_1[1])
abaqus_functions.define_conductivity(aluminum, data_1[2], data_1[3])
abaqus_functions.define_specific_heat(aluminum, data_1[4], data_1[5])
abaqus_functions.define_elasticity(aluminum, data_1[6], data_1[7], data_1[8])
abaqus_functions.define_plasticity(aluminum, data_1[9], data_1[10])
abaqus_functions.define_damage(aluminum, data_1[11])
abaqus_functions.define_inelastic_heat_fraction(aluminum, data_1[12])

# material definition for the dies
steel = abaqus_functions.create_material(model, 'steel')
data_2 = material_data.material_steel()
abaqus_functions.define_density(steel, data_2[0], data_2[1])
abaqus_functions.define_conductivity(steel, data_2[2], data_2[3])
abaqus_functions.define_specific_heat(steel, data_2[4], data_2[5])
abaqus_functions.define_elasticity(steel, data_2[6], data_2[7], data_2[8])

# create and assign sections
abaqus_functions.create_section(model, 'section_aluminum', 'aluminum')
abaqus_functions.assign_section(specimen, set_specimen, 'section_aluminum')
abaqus_functions.create_section(model, 'section_steel', 'steel')
abaqus_functions.assign_section(top_die, set_top_die, 'section_steel')
abaqus_functions.assign_section(bottom_die, set_bottom_die, 'section_steel')

# ----------------------------------------------- ASSEMBLY MODULE -----------------------------------------------------
# create assembly
inst_specimen, inst_bottom_die, inst_top_die = abaqus_functions.create_assembly_press(model, specimen, bottom_die,
                                                                             top_die, h0)
# create reference point
set_rp = abaqus_functions.create_ref_point(model, 'set_rp', h0)

# ----------------------------------------------- STEP MODULE ---------------------------------------------------------
# create step
abaqus_functions.create_temp_disp_expl_step(model, name, time)

# create field and history output
abaqus_functions.delete_automatic_output(model)
# output variables
f_output_all = 'S', 'U', 'NT', 'PE', 'PEEQ', 'COORD', 'DAMAGEC', 'DMICRT', 'CSTRESS', 'CFORCE', 'HFL'
h_output = 'U3', 'RF3', 'V3'  # history output for the reference node
num_interval = 20  # number of intervals for the field output
abaqus_functions.create_field_output_2(model, 'f_output_whole_model', name, f_output_all, num_interval)
num_interval_rp = 100  # number of intervals for the history output of the reference point
abaqus_functions.create_history_output_rp_e(model, 'h_output', 'set_rp', name, h_output, num_interval_rp)

# ----------------------------------------------- INTERACTION MODULE ----------------------------------------------
# create contact between specimen and dies
abaqus_functions.create_contact_property(model, 'contact_property', friction_coefficient, thermal_conductance)
abaqus_functions.create_general_contact_explicit(model, 'general_contact', 'contact_property')

# create kinematic coupling between reference point and top die
top_die_surface = abaqus_functions.create_surfaces(model, inst_top_die, 'top_die_surface', (0, 0, h0 + 0.1))
abaqus_functions.create_kin_coupling(model, 'kinematic_coupling', set_rp, top_die_surface)
```

```python

    # ------------------------------------------- LOAD MODULE ---------------------------------------------------
    # define the initial temperatures
    abaqus_functions.create_predefined_field(model, 'init_temp_top_die', inst_top_die, 'set_top_die',
                                 ambient_temperature)
    abaqus_functions.create_predefined_field_from_output(model, 'init_temp_specimen_bottom_die', inst_specimen,
                                         'set_specimen', odb_abs_path)

    # create amplitude with displacement data
    abaqus_functions.create_amplitude(model, 'amplitude_displacement', amplitude_displacement_data)

    # create boundaries: fix bottom die and apply displacement to top die
    abaqus_functions.create_boundary_fixed(model, 'boundary_fixed', inst_bottom_die, 'set_bottom_die')
    abaqus_functions.create_boundary_displacement(model, 'boundary_displacement', 'amplitude_displacement', set_rp,
                                    name)

    # ------------------------------------------- MESH MODULE ---------------------------------------------------
    # create mesh
    abaqus_functions.create_partitions(specimen, h0)
    abaqus_functions.mesh_control_cylinder(specimen)
    abaqus_functions.create_mesh_3(specimen, seed_size)

    abaqus_functions.create_mesh_3(top_die, seed_size_die)
    abaqus_functions.create_mesh_3(bottom_die, seed_size_die)

    # create node set and output for node set
    abaqus_functions.create_node_set(specimen, 'set_nodes', seed_size, (d0 / 2, 0, 0), (d0 / 2, 0, h0))
    h_output_nodes = 'COOR3', 'NT'
    time_interval = 0.05  # time interval for the history output
    abaqus_functions.create_history_output_e(model, inst_specimen, 'h_output_nodes', 'set_nodes', name, h_output_nodes,
                                 time_interval)

    # ------------------------------------------- JOB MODULE ---------------------------------------------------
    # create job and run simulation
    abaqus_functions.create_job_upsetting_explicit(model_name, name, 6)
    # abaqus_functions.write_input(name, result_path)
    abaqus_functions.submit_job(name, result_path)
    abaqus_functions.wait_for_job(name)

# --------------------------------------------------------------------------------------------------------------

```

## Appendix I: odb_data.py

```
# ------------------------------------------------- SCRIPT INFORMATION ----------------------------------------------------------

# name: odb_data
# function: this script includes general functions to access the Abaqus output database and functions to
evaluate the
# simulations

# ------------------------------------------------- IMPORT ---------------------------------------------------------------------
from abaqus import *
import numpy as np
import os

# ------------------------------------------------- GENERAL FUNCTIONS -----------------------------------------------------------


def open_odb(job_name):
   # function: open the output database and return the odb-object
   # job_name = name of the job
   odb_path = os.getcwd()
   odb = session.openOdb(os.path.join(odb_path, job_name + '.odb'))
   session.viewports['Viewport: 1'].setValues(displayedObject=odb)
   return odb


def close_odb(odb):
   # function: close the output database
   # odb = output database object
   odb.close()


def get_history_output(odb, step_name, region, variable_name):
   # function: access the history output
   # odb = output database object
   history_output = np.array(odb.steps[step_name].historyRegions[region].historyOutputs[
variable_name].data)
   return history_output


def get_column(data, i):
   # function: returns the column i of the history output data
   # data = history output data
   # i = column number
   return [line[:][i] for line in data]


def save_csv(file, mode, data):
   # function: save the specified data in a .csv file
   # file = absolute filepath
   # mode = 'w' for write, 'a' for append
   # data = data that is saved in the file
   with open(file, mode) as f:
      np.savetxt(f, data, delimiter=';')

# ------------------------------------------------- EVALUATE SIMULATION ---------------------------------------------------------


def get_nodal_temperature(job_name, res_path, odb):
   # jobname = name of the job
```

```
    # res_path = absolute path for the result
    # odb = output database object
    # function: save the nodal temperature for a defined set to a .csv file
    # .csv file contains a column for: z-coordinate - time - nodal temperature
    file = os.path.join(res_path, job_name + '.csv')
    with open(file, 'w'):     # create new empty file
        pass
    all_regions = odb.steps[job_name].historyRegions.keys()  # all history region names
    for i, region in enumerate(all_regions):
        if 'Node INSTANCE_SPECIMEN' in region:  # history region name of node set contains this keyword
            history_output = get_history_output(odb, job_name, region, 'COOR3')
            z_coord = get_column(history_output, 1)
            history_output = get_history_output(odb, job_name, region, 'NT11')
            time = get_column(history_output, 0)
            temperature = get_column(history_output, 1)
            node_info = zip(z_coord, time, temperature)
            save_csv(file, 'a', node_info)     # append data to file
    return


def get_nodal_end_temperature(job_name, odb):
    # jobname = name of the job
    # odb = output database object
    # function: save the nodal temperature at the end of a step for a defined node set
    # calculate average temperature of nodes at the end of the step
    all_regions = odb.steps[job_name].historyRegions.keys()  # all history region names
    t = []
    T = []
    for i, region in enumerate(all_regions):
        if 'Node INSTANCE_SPECIMEN' in region:  # history region name of node set contains this keyword
            history_output = get_history_output(odb, job_name, region, 'NT11')
            time = get_column(history_output, 0)
            t.append(time[-1])
            temperature = get_column(history_output, 1)
            T.append(temperature[-1])
    T_end = np.mean(T)
    return T_end


def get_force_displacement(job_name, res_path, odb):
    # jobname = name of the job
    # res_path = absolute path for the result
    # odb = output database object
    # function: save the force displacement data for a reference point
    # .csv file contains a column for: time - displacement in z-direction - reaction force in z-direction
    region = 'Node ASSEMBLY.1'
    history_output = get_history_output(odb, job_name, region, 'RF3')
    t_out = get_column(history_output, 0)  # time
    rf3_out = get_column(history_output, 1)  # reaction force
    history_output = get_history_output(odb, job_name, region, 'U3')
    u3_out = get_column(history_output, 1)  # displacement
    data_ = zip(t_out, u3_out, rf3_out)
    file = os.path.join(res_path, job_name + '_force.csv')
    save_csv(file, 'w', data_)


def evaluate_temperature(job_name, res_path):
    # job_name = name of the job
```

```python
    # res_path = absolute path for the result
    # function: evaluate temperature of a node set and save data to .csv
    odb = open_odb(job_name)
    get_nodal_temperature(job_name, res_path, odb)
    close_odb(odb)


def evaluate_end_temperature(job_name):
    # job_name = name of the job
    # function: evaluate temperature of a node set at the end of a step and calculate average temperature
    odb = open_odb(job_name)
    T_end = get_nodal_end_temperature(job_name, odb)
    close_odb(odb)
    return T_end


def evaluate_upsetting(job_name, res_path):
    # job_name = name of the job
    # res_path = absolute path for the result
    # function: evaluate force-displacement of a reference point and save data to .csv
    odb = open_odb(job_name)
    get_nodal_temperature(job_name, res_path, odb)
    get_force_displacement(job_name, res_path, odb)
    close_odb(odb)

```

## Appendix J: material_data.py

```
# ------------------------------------------- SCRIPT INFORMATION ----------------------------------------------------

# name: material_data.py
# temperature dependent material properties,which are used in the simulations, are defined in this module

# Units: SI-mm
# temperature is defined in degree Celsius

# notes:
# for the density only one value is used (otherwise changes in the function 'define_density' in the abaqus functions
# module needs to be made
# the reference temperature for a property is defined in the _temp variables

# Johnson Cook parameters are entered in the following order:
# jc_parameters = [A, B, n, m, Tm, Tt]
# jc_rate_dependent = [C, epsilon_dot_zero]
# jc_damage = [d1, d2, d3, d4, d5, Tm, Tt, epsilon_dot_zero]

# the order of the material properties needs to be the same for each material: density, density_temp, conductivity,
# conductivity_temp specific_heat, specific_heat_temp, e_modulus, poisson, e_modulus_temp, jc_parameters,
# jc_rate_dependent, jc_damage, inelastic_heat_fraction
# not all properties are defined for silica and steel as they are not needed in the simulations in this application

# literature is listed in the documentation

# ------------------------------------------- SPECIMEN MATERIAL ----------------------------------------------------

def material_aluminum():
    mat_lst = []

    density = 2.7e-09
    density_temp = 20
    conductivity = [191, 197, 204, 211, 218, 225]
    conductivity_temp = [20, 100, 200, 300, 400, 500]
    specific_heat = [911200000, 944000000, 985000000, 1026000000, 1067000000, 1108000000]
    specific_heat_temp = [20, 100, 200, 300, 400, 500]
    e_modulus = [70000, 69300, 67900, 65100, 60200, 54600, 47600, 37800, 28000]
    poisson = [0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33]
    e_modulus_temp = [20, 50, 100, 150, 200, 250, 300, 350, 400]
    jc_parameters = [285, 94, 0.41, 0.9, 588, 25]
    jc_rate_dependent = [0.002, 1]
    jc_damage = [0.0164, 2.245, -2.798, 0.007, 3.65, 582, 25, 1.0]
    inelastic_heat_fraction = 0.9

    mat_lst.append(density)
    mat_lst.append(density_temp)
    mat_lst.append(conductivity)
    mat_lst.append(conductivity_temp)
    mat_lst.append(specific_heat)
    mat_lst.append(specific_heat_temp)
    mat_lst.append(e_modulus)
    mat_lst.append(poisson)
    mat_lst.append(e_modulus_temp)
    mat_lst.append(jc_parameters)
    mat_lst.append(jc_rate_dependent)
    mat_lst.append(jc_damage)
    mat_lst.append(inelastic_heat_fraction)
    return mat_lst


# Material properties of the specimen for the python simulation
# functions to describe the conductivity and the specific heat

def conductivity_function(temperature):
    k = 0.07 * temperature + 190
    return k


def specific_heat_function(temperature):
    c = (0.41 * temperature + 903) * (10 ** 6)
    return c

# ------------------------------------------ MATERIAL OF DIES AND GRIPPER ----------------------------------------


def material_steel():
    mat_lst = []
    density = 7.85e-09
    density_temp = 20.0
    conductivity = [53, 51, 47, 44, 41, 37]
    conductivity_temp = [20, 100, 200, 300, 400, 500]
    specific_heat = [439801760, 487620000, 529760000, 564740000, 605880000, 666500000,
759920000]
    specific_heat_temp = [20, 100, 200, 300, 400, 500, 599]
    e_modulus = [206400, 201600, 198300, 193300, 190600, 186400]
    poisson = [0.271, 0.271, 0.273, 0.275, 0.278, 0.282]
    e_modulus_temp = [50, 100, 150, 200, 250, 295]

    mat_lst.append(density)
    mat_lst.append(density_temp)
    mat_lst.append(conductivity)
    mat_lst.append(conductivity_temp)
    mat_lst.append(specific_heat)
    mat_lst.append(specific_heat_temp)
    mat_lst.append(e_modulus)
    mat_lst.append(poisson)
    mat_lst.append(e_modulus_temp)

    return mat_lst

# ------------------------------------------ FURNACE LINING ------------------------------------------------------


def material_refractory():
    mat_lst = []

    density = 1.82E-09
    density_temp = 20
    conductivity = [1.2, 1.36, 1.51, 1.64, 1.76]
    conductivity_temp = [400, 600, 800, 1000, 1200]
    specific_heat = [915000000, 944000000, 961000000, 969000000, 979000000]
    specific_heat_temp = [400, 600, 800, 1000, 1200]

    mat_lst.append(density)
    mat_lst.append(density_temp)
    mat_lst.append(conductivity)
    mat_lst.append(conductivity_temp)
    mat_lst.append(specific_heat)
    mat_lst.append(specific_heat_temp)
    return mat_lst

# ------------------------------------------------------------------------------------------------------------

```

## Appendix K: abaqus_functions.py

```python
# ---------------------------------------------- SCRIPT INFORMATION ----------------------------------------------------

# name: abaqus_functions
# function: this file contains general functions used to build a FE model in abaqus and run the simulation
# info: the script is divided into sections (one for each Abaqus module)

# ---------------------------------------------- IMPORT -----------------------------------------------------------------
import mesh
from abaqus import *
from abaqusConstants import *
from odbAccess import openOdb
import os

# ---------------------------------------------- MODEL ------------------------------------------------------------------


def model_settings(model_name):
    # FUNCTION: creates a standard/explicit abaqus model, named 'model_name', defines model parameters for the absolute
    # zero and the Boltzmann constant and returns the abaqus model
    # INPUT: model_name     --> name of the abaqus model
    # OUTPUT:               --> abaqus model
    mdb.Model(name=model_name, modelType=STANDARD_EXPLICIT)
    model = mdb.models[model_name]
    model.setValues(absoluteZero=-273.15, stefanBoltzmann=5.67e-11)
    return model


# ---------------------------------------------- PART MODULE ------------------------------------------------------------

def create_cylinder(model, part_name, diameter, height):
    # FUNCTION:             ---> this function creates and returns a part with cylindrical geometry
    # INPUT:    model        ---> abaqus model
    #           part_name    ---> name of the part (string)
    #           diameter     ---> diameter of the cylindrical part
    #           height       ---> dimension in z-direction of the part
    # OUTPUT:   p            ---> abaqus part
    s = model.ConstrainedSketch(name='__profile__', sheetSize=200.0)
    s.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(diameter / 2, 0.0))
    p = model.Part(name=part_name, dimensionality=THREE_D, type=DEFORMABLE_BODY)
    p.BaseSolidExtrude(sketch=s, depth=height)
    del model.sketches['__profile__']
    return p


def create_die(model, part_name, width, length):
    # FUNCTION:             ---> creates and returns a part for the die, the die geometry depends on the workpiece
    #                       dimensions in x,y-directions, the dimension in z-direction is a fixed value,
    #                       the dimensions in x,y-direction are three times the workpiece dimension
    # INPUT:    model        ---> abaqus model
    #           part_name    ---> name of the part (string)
    #           width        ---> dimension of the workpiece in x-direction
    #           length       ---> dimension of the workpiece in y-direction
    # OUTPUT:   p            ---> abaqus part
    width = width * 3
    length = length * 3
    height = 20
    s = model.ConstrainedSketch(name='__profile__', sheetSize=200.0)
    s.rectangle(point1=(-width / 2, -length / 2), point2=(width / 2, length / 2))
    p = model.Part(name=part_name, dimensionality=THREE_D, type=DEFORMABLE_BODY)
    p.BaseSolidExtrude(sketch=s, depth=height)
    return p


def get_dimensions_furnace():
    # FUNCTION:                --> returns the dimensions of the furnace
    # OUTPUT:   dimensions furnace      --> dimensions of the furnace in millimeters
    x_dim_inside = 300  # x-dimension inside the furnace 1 [mm]
    y_dim_inside = 450  # y-dimension inside the furnace 1 [mm]
    z_dim_inside = 240  # z-dimension inside the furnace 1 [mm]
    thickness_walls = 50  # thickness of the furnace walls [mm]
    thickness_bottom_wall = 60  # thickness of the bottom_wall [mm]
    return x_dim_inside, y_dim_inside, z_dim_inside, thickness_walls, thickness_bottom_wall


def create_furnace(model, part_name, width, length, height, thickness, bottom_thickness):
    # FUNCTION:                   --> creates and returns a part for the furnace
    # INPUT:    model              --> abaqus model
    #           part_name          --> name of the part (string)
    #           width           --> inner dimension in x-direction of the furnace
    #           length          --> inner dimension in y-direction of the furnace
    #           height          --> inner dimension in z-direction of the furnace
    #           thickness         --> thickness of the furnace walls
    #           bottom_thickness    --> thickness of the bottom_wall
    # OUTPUT:   p                  --> abaqus part
    # create block with outer dimensions of the furnace
    s = model.ConstrainedSketch(name='__profile__', sheetSize=200.0)
    s.rectangle(point1=(-(width / 2 + thickness), -(length / 2 + thickness)), point2=(width / 2 + thickness,
                                            length / 2 + thickness))
    p = model.Part(name=part_name, dimensionality=THREE_D, type=DEFORMABLE_BODY)
    p.BaseSolidExtrude(sketch=s, depth=height + thickness + bottom_thickness)
    # create datum plane
    plane = p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=bottom_thickness)
    # cut out block with inner dimensions of furnace (width, length, height)
    e, d1 = p.edges, p.datums
    t = p.MakeSketchTransform(sketchPlane=d1[plane.id], sketchUpEdge=e.findAt(
      coordinates=(width / 2 + thickness, (length + 2 * thickness) / 4, 0.0)), sketchPlaneSide=SIDE1,
                 sketchOrientation=RIGHT, origin=(0.0, 0.0, bottom_thickness))
    s1 = model.ConstrainedSketch(name='__profile__', sheetSize=307.21, gridSpacing=7.68, transform=t
)
    p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
    s1.rectangle(point1=(-(width / 2), -(length / 2)), point2=((width / 2), (length / 2)))
    e1, d2 = p.edges, p.datums
    p.CutExtrude(sketchPlane=d2[plane.id], sketchUpEdge=e1.findAt(coordinates=(width / 2 + thickness,
                                         (length + 2 * thickness) / 4, 0.0)),
           sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=height,
flipExtrudeDirection=ON)
    return p


def create_set_all(part, set_name):
    # FUNCTION:                   --> creates a set from a whole part
    # INPUT:    part              --> abaqus part
    #           set name          --> name of the set (string)
    # OUTPUT:   set                --> set
    c = part.cells[:]
    set_all = part.Set(cells=c, name=set_name)
    return set_all


# ---------------------------------------------- PROPERTY MODULE ------------------------------------------------------------
def create_material(model, material_name):
    # FUNCTION:                 ---> creates a new material
    # INPUT:    model            ---> abaqus model
    #           material name    ---> name of the material (string)
    # OUTPUT:   material         ---> material
    material = model.Material(name=material_name)
    return material


def define_density(material, density, reference_temperature):
    material.Density(table=((density, reference_temperature),), temperatureDependency=ON)


def define_elasticity(material, e_modulus, poisson, temperature):
    values = list(zip(e_modulus, poisson, temperature))
    table_values = []
    for i in range(len(e_modulus)):
        table_values.append(values[i])
    material.Elastic(table=table_values, temperatureDependency=ON)


def define_conductivity(material, conductivity, temperature):
    values = list(zip(conductivity, temperature))
    table_values = []
    for i in range(len(conductivity)):
        table_values.append(values[i])
    material.Conductivity(table=table_values, temperatureDependency=ON)


def define_specific_heat(material, specific_heat, temperature):
    values = list(zip(specific_heat, temperature))
    table_values = []
    for i in range(len(specific_heat)):
        table_values.append(values[i])
    material.SpecificHeat(table=table_values, temperatureDependency=ON, law=CONSTANTPRESSURE)


def define_expansion(material, expansion, temperature):
    values = list(zip(expansion, temperature))
    table_values = []
    for i in range(len(expansion)):
        table_values.append(values[i])
    material.Expansion(table=table_values, temperatureDependency=ON)


def define_plasticity(material, jc_params, jc_rate_dep):
    material.Plastic(hardening=JOHNSON_COOK, table=(jc_params,))
    material.plastic.RateDependent(type=JOHNSON_COOK, table=(jc_rate_dep,))


def define_damage(material, jc_damage_params):
    material.JohnsonCookDamageInitiation(table=(jc_damage_params,))
```

```


def define_inelastic_heat_fraction(material, fraction):
    material.InelasticHeatFraction(fraction)


def create_section(model, section_name, material):
    model.HomogeneousSolidSection(name=section_name, material=material, thickness=None)


def assign_section(part, set_all, section_name):
    part.SectionAssignment(region=set_all, sectionName=section_name, offset=0.0, offsetType=MIDDLE_SURFACE,
                           offsetField='', thicknessAssignment=FROM_SECTION)


# ------------------------------------------- ASSEMBLY MODULE ------------------------------------------------------
def create_assembly_press(model, part1, part2, part3, height_workpiece):
    # FUNCTION:                ---> creates an assembly for the press and positions the part
    #                          distance between dies = height_workpiece + 0.1
    # INPUT:   model           ---> abaqus model
    #          part            ---> parts of the assembly (part1=workpiece, part2=top die, part3=bottom die)
    #          height_workpiece  ---> z-dimension of the workpiece
    # OUTPUT:  instance        ---> instances of the assembly (instance1=workpiece, instance2=top die,
    #                          instance3=bottom die)
    a = model.rootAssembly
    a.DatumCsysByDefault(CARTESIAN)
    instance1 = a.Instance(dependent=ON, name='instance_specimen', part=part1)
    instance2 = a.Instance(dependent=ON, name='instance_bottom_die', part=part2)
    instance3 = a.Instance(dependent=ON, name='instance_top_die', part=part3)
    a.translate(instanceList=('instance_bottom_die',), vector=(0.0, 0.0, -20))
    # -20 is equal to the height of the bottom die
    a.translate(instanceList=('instance_top_die',), vector=(0.0, 0.0, height_workpiece + 0.1))
    return instance1, instance2, instance3


def create_assembly_2parts(model, part1, part2, part1_name, part2_name, translation):
    # FUNCTION:                ---> creates an assembly with two parts, positions the second part
    # INPUT:   model           ---> abaqus model
    #          part            ---> parts of the assembly (part1=workpiece)
    #          translation     ---> positions the second part in this direction along the z-axis
    # OUTPUT:  instance        ---> instances of the assembly (instance1=workpiece)
    a = model.rootAssembly
    a.DatumCsysByDefault(CARTESIAN)
    instance1 = a.Instance(dependent=ON, name=part1_name, part=part1)
    instance2 = a.Instance(dependent=ON, name=part2_name, part=part2)
    a.translate(instanceList=(part2_name,), vector=(0.0, 0.0, -translation))
    return instance1, instance2


def create_assembly_furnace(model, part1, part2, translation):
    # FUNCTION:                ---> creates an assembly with the furnace and the specimen
    # INPUT:   model           ---> abaqus model
    #          part 1          ---> specimen
    #          part 2          ---> furnace
    #          translation     ---> translation of the furnace
    a = model.rootAssembly
    a.DatumCsysByDefault(CARTESIAN)
    instance1 = a.Instance(dependent=ON, name='instance_specimen', part=part1)
    instance2 = a.Instance(dependent=ON, name='instance_furnace', part=part2)
    a.translate(instanceList=('instance_furnace',), vector=(0.0, 0.0, -translation))
    return instance1, instance2


def create_ref_point(model, set_name, height_workpiece):
    # FUNCTION:                 ---> creates a reference point at the z-position: height_workpiece + 0.1
    #                           (= position of contact surface of top die)
    # INPUT:   model             ---> abaqus model
    #          set_name          ---> name of the set (string)
    #          height_workpiece   ---> z-dimension of the workpiece
    # OUTPUT:  set               ---> set including the reference point
    a = model.rootAssembly
    rp = a.ReferencePoint(point=(0.0, 0.0, height_workpiece + 0.1))
    return a.Set(name=set_name, referencePoints=(a.referencePoints[rp.id],))


def create_surfaces(model, instance, surface_name, *coordinates):
    # FUNCTION:                 ---> creates surfaces, each surface is selected by x,y,z coordinates of
    #                           a point in the middle of the surface
    # INPUT:   model             ---> abaqus model
    #          instance          ---> abaqus instance
    #          surface_name       ---> name of the surfaces (string)
    # OUTPUT:  created_surface    ---> abaqus surfaces
    a = model.rootAssembly
    f = instance.faces
    surface = ()
    for coordinate in coordinates:
        x, y, z = coordinate
        face = f.findAt((x, y, z), )
        surface = surface + (f[face.index:face.index + 1],)
    created_surface = a.Surface(side1Faces=surface, name=surface_name)
    return created_surface


def create_set_faces(model, set_name, instance, *coordinates):
    # FUNCTION:                 ---> creates a set with faces, each face is selected by x,y,z coordinates of
    #                           a point in the middle of the face
    # INPUT:   model             ---> abaqus model
    #          set_name          ---> name of the set (string)
    #          instance          ---> abaqus instance
    # OUTPUT:  set               ---> abaqus set including the selected faces
    a = model.rootAssembly
    f = instance.faces
    selected_faces = ()
    for coordinate in coordinates:
        x, y, z = coordinate
        face = f.findAt((x, y, z), )
        selected_faces = selected_faces + (f[face.index:face.index + 1],)
    return a.Set(faces=selected_faces, name=set_name)


# ---------------------------------------------- STEP MODULE ----------------------------------------------------------
def create_temp_disp_expl_step(model, step_name, time_period):
    # creates a dynamic temp-displ explicit step named 'step_name', whereas time_period is the step
time
    model.TempDisplacementDynamicsStep(name=step_name, previous='Initial', timePeriod=
```

```python
time_period, improvedDtMethod=ON)


def coupled_tep_displ_step(model, step_name, time_period):
    # creates a coupled temperature displacement step named 'step_name', whereas time_period is the step time
    model.CoupledTempDisplacementStep(name=step_name, previous='Initial', timePeriod=time_period, maxNumInc=500,
                                      initialInc=0.005, minInc=1e-08, maxInc=0.05, deltmx=100, nlgeom=ON)


def create_heat_transfer_step(model, step_name, previous_step_name, step_time):
    # creates a heat transfer step named 'step_name' whereas step_time is the time of the step
    model.HeatTransferStep(deltmx=10.0, end=1e-07, initialInc=0.001, maxInc=2.0, maxNumInc=10000, minInc=1e-07,
                           name=step_name, previous=previous_step_name, timePeriod=step_time)


def delete_automatic_output(model):
    # deletes the automatic field and history output that is defined in a new model database
    del model.fieldOutputRequests['F-Output-1']
    del model.historyOutputRequests['H-Output-1']


def delete_f_output1(model):
    # deletes the automatic field output that is defined in a new model database
    del model.fieldOutputRequests['F-Output-1']


def create_field_output(model, f_output_name, name_of_step, output_variables, time_interval):
    # creates a field output for the whole model with the name 'f_output_name' for the step 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    model.FieldOutputRequest(createStepName=name_of_step, name=f_output_name, timeInterval=time_interval,
                             timeMarks=OFF, variables=output_variables)


def create_field_output_2(model, f_output_name, name_of_step, output_variables, num_interval):
    # creates a field output for the whole model with the name 'f_output_name' for the step 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    model.FieldOutputRequest(createStepName=name_of_step, name=f_output_name, numIntervals=num_interval,
                             timeMarks=OFF, variables=output_variables)


def create_field_output_set(model, instance, f_output_name, set_name, step_name, output_variables, num_interval):
    # creates a field output with the name 'f_output_name' for the set of an abaqus instance
    # the field output is created for the step named 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    regionDef = instance.sets[set_name]
    model.FieldOutputRequest(name=f_output_name, createStepName=step_name, variables=output_variables,
                             numIntervals=num_interval, region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)
```

```


def create_history_output_rp(model, h_output_name, set_name, step_name, output_variables, num_interval):
    # creates a history output with the name 'h_output_name' for a set of the assembly named 'set_name'
    # the history output is created for the step named 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    regionDef = model.rootAssembly.sets[set_name]
    model.HistoryOutputRequest(name=h_output_name, createStepName=step_name, variables=output_variables,
                               numIntervals=num_interval, region=regionDef, sectionPoints=DEFAULT, timeMarks=OFF,
                               rebar=EXCLUDE)


def create_history_output_rp_e(model, h_output_name, set_name, step_name, output_variables, num_interval):
    # creates a history output with the name 'h_output_name' for a set of the assembly named 'set_name'
    # for the explicit simulation
    # the history output is created for the step named 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    regionDef = model.rootAssembly.sets[set_name]
    model.HistoryOutputRequest(name=h_output_name, createStepName=step_name, variables=output_variables,
                               numIntervals=num_interval, region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)


def create_history_output(model, instance, h_output_name, set_name, step_name, output_variables, time_interval):
    # creates a history output with the name 'h_output_name' for a set of an instance named 'set_name'
    # the history output is created for the step named 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    regionDef = instance.sets[set_name]
    model.HistoryOutputRequest(name=h_output_name, createStepName=step_name, variables=output_variables,
                               timeInterval=time_interval, region=regionDef, sectionPoints=DEFAULT, timeMarks=OFF,
                               rebar=EXCLUDE)


def create_history_output_e(model, instance, h_output_name, set_name, step_name, output_variables, time_interval):
    # creates a history output with the name 'h_output_name' for a set of an instance named 'set_name'
    # for the explicit simulation
    # the history output is created for the step named 'step_name'
    # output_variables defines all the variables for the output e.g. 'S', 'RF'
    # time interval specifies the output interval
    regionDef = instance.sets[set_name]
    model.HistoryOutputRequest(name=h_output_name, createStepName=step_name, variables=output_variables,
                               timeInterval=time_interval, region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)
```

```


# ---------------------------------------------- INTERACTION MODULE --------------------------------------------------
def create_kin_coupling(model, coupling_name, set_ref_point, slave_surface):
    # creates a kinematic coupling named 'coupling_name' between a reference point and a slave surface
    # all degrees of freedom are constrained
    model.Coupling(name=coupling_name, controlPoint=set_ref_point, surface=slave_surface, influenceRadius=WHOLE_SURFACE,
                   couplingType=KINEMATIC, localCsys=None, u1=ON, u2=ON, u3=ON, ur1=ON, ur2=ON, ur3=ON)


def create_contact_property(model, interaction_property_name, friction_coefficient, thermal_conductance):
    # creates a contact property including friction and thermal contact conductance
    model.ContactProperty(interaction_property_name)
    ip = model.interactionProperties[interaction_property_name]
    ip.NormalBehavior(pressureOverclosure=HARD, allowSeparation=ON, constraintEnforcementMethod=DEFAULT)
    ip.TangentialBehavior(formulation=PENALTY, directionality=ISOTROPIC, slipRateDependency=OFF, pressureDependency=OFF,
                          temperatureDependency=OFF, dependencies=0, table=((friction_coefficient,),),
                          shearStressLimit=None,
                          maximumElasticSlip=FRACTION, fraction=0.005, elasticSlipStiffness=None)
    ip.ThermalConductance(definition=TABULAR, clearanceDependency=ON, pressureDependency=OFF,
                          temperatureDependencyC=OFF,
                          massFlowRateDependencyC=OFF, dependenciesC=0, clearanceDepTable=thermal_conductance)


def create_contact_property_thermal(model, interaction_property_name, thermal_conductance):
    # creates a contact property including thermal contact conductance
    model.ContactProperty(interaction_property_name)
    ip = model.interactionProperties[interaction_property_name]
    ip.ThermalConductance(definition=TABULAR, clearanceDependency=ON, pressureDependency=OFF,
                          temperatureDependencyC=OFF,
                          massFlowRateDependencyC=OFF, dependenciesC=0, clearanceDepTable=thermal_conductance)


def create_general_contact_explicit(model, name, property_name):
    # create a general contact interaction for an explicit simulation
    # name = name of the interaction
    # property_name = name of the interaction property
    model.ContactExp(name=name, createStepName='Initial')
    model.interactions[name].includedPairs.setValuesInStep(stepName='Initial', useAllstar=ON)
    model.interactions[name].contactPropertyAssignments.appendInStep(stepName='Initial', assignments=((GLOBAL, SELF,
                                                                                                      property_name),))


def create_general_contact(model, name, property_name):
    # create a general contact interaction
    # name = name of the interaction
    # property_name = name of the interaction property
    model.ContactStd(name=name, createStepName='Initial')
    model.interactions[name].includedPairs.setValuesInStep(stepName='Initial', useAllstar=ON)
    model.interactions[name].contactPropertyAssignments.appendInStep(stepName='Initial', assignments=((GLOBAL, SELF,
                                                                                                        property_name),))


def create_contact_interaction(model, int_name, master_surface, slave_surface, int_property_name):
    # int_name = interaction name
    # int_property_name = name of the interaction property that should be used
    region1 = master_surface
    region2 = slave_surface
    model.SurfaceToSurfaceContactStd(name=int_name, createStepName='Initial', master=region1, slave=region2,
                                     sliding=FINITE, thickness=ON, interactionProperty=int_property_name,
                                     adjustMethod=NONE, initialClearance=OMIT, datumAxis=None, clearanceRegion=None)


def create_contact_interaction_expl(model, int_name, master_surface, slave_surface, int_property_name):
    # int_name = interaction name
    # int_property_name = name of the interaction property that should be used
    region1 = master_surface
    region2 = slave_surface
    model.SurfaceToSurfaceContactExp(name=int_name, createStepName='Initial', master=region1, slave=region2,
                                     mechanicalConstraint=PENALTY, sliding=FINITE,
                                     interactionProperty=int_property_name, initialClearance=OMIT,
                                     datumAxis=None, clearanceRegion=None)


def create_radiation_to_ambient(model, name, surface, step_name, ambient_temperature, emissivity):
    # create radiation to ambient with constant temperature
    region = surface
    model.RadiationToAmbient(name=name, createStepName=step_name, surface=region, radiationType=AMBIENT,
                             distributionType=UNIFORM, field='', emissivity=emissivity,
                             ambientTemperature=ambient_temperature, ambientTemperatureAmp='')


def create_radiation_to_var_ambient(model, name, surface, step_name, amplitude_name, emissivity):
    # create radiation to ambient with variable temperature amplitude
    region = surface
    model.RadiationToAmbient(name=name, createStepName=step_name, surface=region, radiationType=AMBIENT,
                             distributionType=UNIFORM, field='', emissivity=emissivity, ambientTemperature=1.0,
                             ambientTemperatureAmp=amplitude_name)


def create_convection_var(model, name, surface, step_name, amplitude_name, film_coefficient):
    # create convection interaction to ambient with variable temperature amplitude
    region = surface
    model.FilmCondition(name=name, createStepName=step_name, surface=region, definition=EMBEDDED_COEFF,
                        filmCoeff=film_coefficient, filmCoeffAmplitude='', sinkTemperature=1.0,
                        sinkAmplitude=amplitude_name, sinkDistributionType=UNIFORM, sinkFieldName='')


```

```python
def create_convection(model, name, surface, step_name, film_coefficient, ambient_temperature):
    # create convection interaction to ambient with constant temperature
    region = surface
    model.FilmCondition(name=name, createStepName=step_name, surface=region, definition=EMBEDDED_COEFF,
                        filmCoeff=film_coefficient, filmCoeffAmplitude='', sinkTemperature=ambient_temperature,
                        sinkAmplitude='', sinkDistributionType=UNIFORM, sinkFieldName='')


# ---------------------------------------------- LOAD MODULE ----------------------------------------------------------
def create_amplitude(model, amplitude_name, amplitude_data):
    # creates a tabular amplitude named 'amplitude_name' using tabular data: amplitude_data
    model.TabularAmplitude(name=amplitude_name, timeSpan=STEP, smooth=SOLVER_DEFAULT, data=amplitude_data)


def create_predefined_field(model, field_name, instance, set_name, temperature):
    # defines an initial predefined temperature field named 'field_name' with a constant temperature to the set of an
    # instance named 'set_name'
    region = instance.sets[set_name]
    model.Temperature(name=field_name, createStepName='Initial', region=region, distributionType=UNIFORM,
                      crossSectionDistribution=CONSTANT_THROUGH_THICKNESS, magnitudes=(temperature,))


def create_predefined_field_from_output(model, field_name, instance, set_name, abs_file_path):
    # defines an initial predefined temperature field named 'field_name' to the set named 'set_name' of an instance
    # the temperature field is defined by a previous simulation; this function reads the temperature field of the last
    # increment of the last step of an odb file specified by 'abs_file_path'
    region = instance.sets[set_name]
    odb = openOdb(abs_file_path)
    last_step = odb.steps.values()[-1]
    nr_last_step = last_step.number
    nr_inc = last_step.frames[-1].incrementNumber
    odb.close()
    model.Temperature(name=field_name, createStepName='Initial', distributionType=FROM_FILE, fileName=abs_file_path,
                      beginStep=nr_last_step, beginIncrement=nr_inc, endStep=None, endIncrement=None, interpolate=OFF,
                      absoluteExteriorTolerance=0.0, exteriorTolerance=0.05)
    model.predefinedFields[field_name].setValues(region=region)  # todo: is a region definition needed?
    # INFO: interpolate = ON for incompatible meshes interpolate = OFF for compatible meshes


def create_boundary_fixed(model, boundary_name, instance, set_name):
    # creates a boundary named 'boundary_name' in the initial step that constrains all degrees of freedom for the
    # defined set named 'set_name' of an instance
    region = instance.sets[set_name]
    model.EncastreBC(name=boundary_name, createStepName='Initial', region=region, localCsys=None)


def create_boundary_displacement(model, boundary_name, amplitude_name, set_ref_point,
step_name):
    # creates a displacement boundary condition named 'boundary_name' in the step named 'step_name', the region is
    # defined by the set set_ref_point that contains a reference point; translational displacement is applied in z-
    # direction of the model via a tabular amplitude named 'amplitude_name' that specifies the time - displacement
    # values
    model.DisplacementBC(name=boundary_name, createStepName=step_name, region=set_ref_point, u1=0.0, u2=0.0, u3=-1.0,
                         ur1=0.0, ur2=0.0, ur3=0.0, amplitude=amplitude_name, fixed=OFF, distributionType=UNIFORM,
                         fieldName='', localCsys=None)


def create_boundary_temperature(model, boundary_name, set_faces, step_name, amplitude_name):
    # applies a temperature boundary condition named 'boundary_name' to a set containing faces 'set_faces' in the step
    # 'step_name'; a tabular amplitude 'amplitude_name' specifies the time - temperature data
    model.TemperatureBC(name=boundary_name, createStepName=step_name, region=set_faces, fixed=OFF,
                        distributionType=UNIFORM, fieldName='', magnitude=1.0, amplitude=amplitude_name)


# ------------------------------------------- MESH MODULE -------------------------------------------------------
def create_mesh_1(part, seeds):
    # seed specifies the element size of the mesh that is created;
    # elements for heat transfer
    part.seedPart(size=seeds, deviationFactor=0.1, minSizeFactor=0.1)
    elemType1 = mesh.ElemType(elemCode=DC3D8, elemLibrary=STANDARD)
    elemType2 = mesh.ElemType(elemCode=DC3D6, elemLibrary=STANDARD)
    elemType3 = mesh.ElemType(elemCode=DC3D4, elemLibrary=STANDARD)
    cells = part.cells[:]
    pickedRegions = (cells,)
    part.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2, elemType3))
    part.generateMesh()


def create_mesh_2(part, seeds):
    # seed specifies the element size of the mesh that is created;
    # elements coupled temperature displacement (implicit)
    part.seedPart(size=seeds, deviationFactor=0.1, minSizeFactor=0.1)
    elemType1 = mesh.ElemType(elemCode=C3D8RT, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
                              distortionControl=DEFAULT)
    elemType2 = mesh.ElemType(elemCode=C3D6T, elemLibrary=STANDARD)
    elemType3 = mesh.ElemType(elemCode=C3D4T, elemLibrary=STANDARD)
    cells = part.cells[:]
    pickedRegions = (cells,)
    part.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2, elemType3))
    part.generateMesh()


def create_mesh_3(part, seeds):
    # seed specifies the element size of the mesh that is created;
    # elements for coupled temperature displacement (explicit)
    part.seedPart(size=seeds, deviationFactor=0.1, minSizeFactor=0.1)
    elemType1 = mesh.ElemType(elemCode=C3D8RT, elemLibrary=EXPLICIT, secondOrderAccuracy=OFF,
                          distortionControl=DEFAULT)
    elemType2 = mesh.ElemType(elemCode=C3D6T, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4T, elemLibrary=EXPLICIT)
    cells = part.cells[:]
    pickedRegions = (cells,)
    part.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2, elemType3))
    part.generateMesh()


def create_partitions(part, height):
    # INPUT: part    --> abaqus part (specimen)
    #        height --> inital height of the specimen
    # partitions are defined using datum planes; partitions are created along the yz-plane, the xz-plane and at an
    # offset of height/2 to the xy plane; 'height' is the height of the workpiece
    p = part
    c = p.cells[:]
    yz_plane = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset=0.0)
    xz_plane = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)
    xy_plane = p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=height / 2)
    p.PartitionCellByDatumPlane(datumPlane=p.datums[yz_plane.id], cells=c)
    c = p.cells[:]
    p.PartitionCellByDatumPlane(datumPlane=p.datums[xz_plane.id], cells=c)
    c = p.cells[:]
    p.PartitionCellByDatumPlane(datumPlane=p.datums[xy_plane.id], cells=c)


def mesh_control_cylinder(part):
    # applies mesh controls to the defined part;
    # default settings are elemShape: HEX, technique: STRUCTURED
    c = part.cells[:]
    part.setMeshControls(regions=c, elemShape=HEX_DOMINATED, technique=SWEEP, algorithm=ADVANCING_FRONT)


def create_partitions_furnace(part, x, y, z, t):
    # creates partitions of the part 'furnace'
    # x, y, z are the inner dimensions of the furnace in the respective directions; t is the thickness of the bottom
    # wall of the furnace
    c = part.cells[:]
    v = part.vertices
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(-x / 2, -y / 2, t)),
                                         point2=v.findAt(coordinates=(x / 2, -y / 2, t)),
                                         point3=v.findAt(coordinates=(x / 2, -y / 2, z + t)), cells=c)
    c = part.cells[:]
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(-x / 2, y / 2, t)),
                                         point2=v.findAt(coordinates=(x / 2, y / 2, t)),
                                         point3=v.findAt(coordinates=(x / 2, y / 2, z + t)), cells=c)
    c = part.cells[:]
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(x / 2, -y / 2, t)),
                                         point2=v.findAt(coordinates=(x / 2, y / 2, t)),
                                         point3=v.findAt(coordinates=(x / 2, y / 2, z + t)), cells=c)
    c = part.cells[:]
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(-x / 2, -y / 2, t)),
                                         point2=v.findAt(coordinates=(-x / 2, y / 2, t)),
                                         point3=v.findAt(coordinates=(-x / 2, y / 2, z + t)), cells=c)
    c = part.cells[:]
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(-x / 2, -y / 2, t)),
                                         point2=v.findAt(coordinates=(x / 2, -y / 2, t)),
                                         point3=v.findAt(coordinates=(x / 2, y / 2, t)), cells=c)
    c = part.cells[:]
    part.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(-x / 2, -y / 2, z + t)),
                                         point2=v.findAt(coordinates=(x / 2, -y / 2, z + t)),
                                         point3=v.findAt(coordinates=(x / 2, y / 2, z + t)), cells=c)


def create_node_set(part, set_name, seed_size, point1, point2):
    # INPUT: point1     --> coordinates (x,y,z) of starting point
    #        point2     --> coordinates (x,y,z) of end point
    # FUNCTION: select nodes along a line (in positive x, y, or z- direction) and saves them in a set named 'set_name';
    # the line is defined x,y,z coordinates of a starting point 'point1' and a end point 'point2'; 'seed_size' refers to
    # the element size of the mesh, which is used as a tolerance value for the bounding box
    tol = seed_size / 2
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    n = part.nodes
    nodes = n.getByBoundingBox(x1 - tol, y1 - tol, z1 - tol, x2 + tol, y2 + tol, z2 + tol)
    part.Set(nodes=nodes, name=set_name)


# ------------------------------------------- JOB MODULE -----------------------------------------------------------
def create_job_upsetting_explicit(model_name, job_name, nr_cpu):
    # nr_cpu = number of CPUs
    # explicit upsetting
    # creates a job named 'job_name' for the upsetting model named 'model_name'; parallelization is used, full precision
    mdb.models[model_name].rootAssembly.regenerate()
    mdb.Job(name=job_name, model=model_name, description='', type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
        queue=None, memory=90, memoryUnits=PERCENTAGE, explicitPrecision=DOUBLE_PLUS_PACK,
        nodalOutputPrecision=FULL, echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
        userSubroutine='', scratch='', resultsFormat=ODB, parallelizationMethodExplicit=DOMAIN, numDomains=nr_cpu,
        activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=nr_cpu)


def create_job_upsetting(model_name, job_name, nr_cpu):
    # nr_cpu = number of CPUs
    # implicit upsetting
    # creates a job named 'job_name' for the upsetting model named 'model_name';
    mdb.models[model_name].rootAssembly.regenerate()
    mdb.Job(name=job_name, model=model_name, description='', type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
        queue=None, memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
        nodalOutputPrecision=SINGLE, echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
        userSubroutine='', scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=nr_cpu,
        numDomains=nr_cpu, numGPUs=0)


```

```python
def create_job_heat_transfer(model_name, job_name, nr_cpu):
    # creates a job named 'job_name' for a heat transfer model named 'model_name'
    # nr_cpu = number of CPUs
    mdb.Job(name=job_name, model=model_name, description='', type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
            queue=None, memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
            nodalOutputPrecision=SINGLE, echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
            userSubroutine='', scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=nr_cpu,
            numDomains=nr_cpu, numGPUs=0)


def submit_job(job_name, result_path):
    # submits the job named 'job_name' and waits for the completion of the job; the working directory is changed to
    # 'result_path' and all the files are saved in this directory
    os.chdir(result_path)
    mdb.jobs[job_name].submit(consistencyChecking=OFF)


def wait_for_job(job_name):
    # submits the job named 'job_name' and waits for the completion of the job; the working directory is changed to
    # 'result_path' and all the files are saved in this directory
    mdb.jobs[job_name].waitForCompletion()


def write_input(job_name, result_path):
    # writes the input file for the job named 'job_name'; the working directory is changed to 'result_path' the input
    # file is saved in this directory
    os.chdir(result_path)
    mdb.jobs[job_name].writeInput()

# ---------------------------------------------------------------------------------------------------------------

```

## Appendix L: Documentation of experiments

| Experiment 1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Setting: s1, $d_0 = 10\ mm, h_0 = 15\ mm, T_F = 300\ °C,\ t_t = 4\ s, \Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_4 | 14.81 | 10.00 | | TestNr_10 | 15.03 | 10.00 | |
| TestNr_8 | 15.13 | 10.16 | | TestNr_11 | 15.11 | 10.16 | |
| TestNr_9 | 14.78 | 9.79 | | TestNr_12 | 15.02 | 9.79 | |
| Setting: s2, $d_0 = 10\ mm, h_0 = 15\ mm,\ T_F = 300\ °C,\ t_t = 4\ s, \Delta h = 8\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_13 | 14.80 | 7.17 | X | TestNr_16 | 15.04 | 7.54 | X |
| TestNr_14 | 15.14 | 7.42 | X | TestNr_17 | 14.99 | 7.52 | X |
| TestNr_15 | 14.98 | 7.55 | X | TestNr_18 * | 14.85 | 7.55 | X |
| Setting: s3, $d_0 = 20\ mm, h_0 = 30\ mm,\ T_F = 300\ °C,\ t_t = 4\ s, \Delta h = 15\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_19 | 30.80 | 15.34 | | TestNr_22 | 30.09 | 15.35 | |
| TestNr_20 | 30.09 | 15.40 | X | TestNr_23 | 30.08 | 15.42 | |
| TestNr_21 | 30.07 | 15.40 | | TestNr_24 | 30.13 | 15.34 | X |
| Setting: s4, $d_0 = 20\ mm, h_0 = 30\ mm,\ T_F = 300\ °C,\ t_t = 7\ s, \Delta h = 20\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_31 | 30.11 | 10.56 | | TestNr_32 | 30.40 | 10.71 | |
| TestNr_26 | 30.12 | 10.82 | X | TestNr_29 | 30.08 | 10.63 | |
| TestNr_27 | 30.03 | 10.70 | | TestNr_30 | 30.15 | 10.67 | X |
| Setting: s5, $d_0 = 10\ mm, h_0 = 15\ mm, T_F = 500\ °C,\ t_t = 7\ s, \Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_33 | 15.18 | 9.81 | | TestNr_36 | 15.18 | 9.77 | |
| TestNr_34 | 14.88 | 9.57 | | TestNr_37 | 15.18 | 9.58 | |
| TestNr_35 | 15.12 | 9.70 | | TestNr_38 | 15.16 | 9.67 | |
| Setting: s6, $d_0 = 10\ mm, h_0 = 15\ mm, T_F = 500\ °C,\ t_t = 4\ s, \Delta h = 8\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_39 | 15.19 | 6.99 | | TestNr_42 | 15.06 | 6.97 | |
| TestNr_40 | 15.01 | 6.92 | | TestNr_43 | 15.18 | 6.83 | |
| TestNr_41 | 15.03 | 7.00 | | TestNr_44 | 15.16 | 6.86 | |
| Setting: s7, $d_0 = 20\ mm, h_0 = 30\ mm, T_F = 500\ °C,\ t_t = 7\ s, \Delta h = 15\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_45 | 30.10 | 14.84 | | TestNr_48 | 30.04 | 14.95 | X |
| TestNr_46 | 30.06 | 14.83 | X | TestNr_49 | 30.00 | 14.79 | X |
| TestNr_47 | 30.42 | 15.02 | X | TestNr_50 | 30.01 | 14.71 | X |
| Setting: s8, $d_0 = 20\ mm, h_0 = 30\ mm, T_F = 500\ °C,\ t_t = 4\ s, \Delta h = 20\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_51 | 30.00 | 10.20 | | TestNr_54 | 30.14 | 10.24 | |
| TestNr_52 | 29.20 | 10.04 | | TestNr_55 | 30.00 | 10.18 | |
| TestNr_53 | 30.04 | 10.23 | X | TestNr_56 | 30.20 | 10.15 | X |

* Not valid; excluded
Visible cracks, that occurred during the test, are marked with X.

| Experiment 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Setting: s1, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 300\ °C$, $t_t = 4\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_12 | 14.87 | 9.61 | | TestNr_14 | 15.02 | 9.48 | |
| TestNr_13 | 14.98 | 9.47 | | TestNr_15 | 14.95 | 9.46 | |
| Setting: s2, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 300\ °C$, $t_t = 7\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_16 | 14.75 | 9.40 | | TestNr_18 | 15.05 | 9.39 | |
| TestNr_17 | 14.93 | 9.58 | | TestNr_19 | 14.81 | 9.48 | |
| Setting: s3, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 400\ °C$, $t_t = 4\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_20 | 15.13 | 9.27 | | TestNr_22 | 14.98 | 9.28 | |
| TestNr_21 | 15.13 | 9.52 | | TestNr_23 | 14.95 | 9.47 | |
| Setting: s4, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 400\ °C$, $t_t = 7\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_24 | 15.09 | 9.46 | | TestNr_26 | 15.05 | 9.38 | |
| TestNr_25 | 14.95 | 9.43 | | TestNr_27 | 15.02 | 9.44 | |
| Setting: s5, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 500\ °C$, $t_t = 4\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_28 | 14.96 | 9.46 | | TestNr_30 | 15.04 | 9.29 | |
| TestNr_29 | 14.95 | 9.37 | | TestNr_31 | 15.16 | 9.25 | |
| Setting: s6, $d_0 = 10\ mm$, $h_0 = 15\ mm$, $T_F = 500\ °C$, $t_t = 7\ s$, $\Delta h = 5\ mm$ | | | | | | | |
| Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks | Measurement | $h_0[mm]$ | $h_1[mm]$ | Cracks |
| TestNr_32 | 14.88 | 9.28 | | TestNr_34 | 15.01 | 9.40 | |
| TestNr_33 | 15.08 | 9.41 | | TestNr_35 | 14.96 | 9.28 | |

Visible cracks, that occurred during the test, are marked with X.