Development of Open-Source Software for Repeatable Processing of Mechanical Test Data

Presented by:
Mr. Daniel Slater

Supervisors:
Mr. E. Ismail, Dr. S. George, Dr. B. Alheit

Submitted to the Department of Mechanical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Master of Science degree in Mechanical Engineering.

July 2023

University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed: Signed by candidate

Acknowledgements

My parents,
My anchor in the sea.
Their love and care,
Always with me.

My supervisors,
Ben, Ernesto and Sarah,
a trio most sage.

CME and CERECAM,
A community of grace.
A fellowship of scholars
In this place.

NRF and Hulamin,
Funded my academic quest,
Giving me the chance
To do my best.

So here’s to you,
My family and friends,
Thank you for your support.
You are the reason,
I’m here to report.

Abstract

Experiments that test for the mechanical properties of materials typically generate large amounts of raw data that must be cleaned and processed before meaningful analysis can be performed. Manual processing of this data is often time-consuming and susceptible to errors. Furthermore, the necessary processing steps are usually similar for various datasets, given that mechanical testing procedures are standardised. These issues were addressed in this project through the development of a software toolkit for automated processing and repeatable analysis of mechanical test data.

The toolkit, named Paramaterial, was developed as a pip installable Python package, and is designed for use in Jupyter Notebooks. The usage of Jupyter Notebooks makes the steps that a user took in processing and analysing data explicit, thus providing traceability and repeatability. The functionalities of Paramaterial were demonstrated by processing and analysing several datasets sourced from the literature and from the Centre for Material Engineering (CME) at the University of Cape Town (UCT).

The example dataset gathered for the demonstrations consists of 100 uniaxial tensile tests, and 56 plane-strain tension tests on aluminium AA6061, as well as 70 uniaxial compression tests, and 40 plane-strain compression tests on aluminium AA3104. The Jupyter Notebooks containing the code for these demonstrations serve as tutorials for future users of the toolkit. Code documentation and user manuals have also been provided, making the software readily available to be used for improving the quality and quantity of processed experimental data in the field.

Table of Contents

Acknowledgements **3**

Abstract **4**

Contents **5**

List of Figures **8**

List of Tables **10**

List of Listings **11**

List of Algorithms **12**

Acronyms **13**

1 Introduction **15**

1.1 Background . . . 15
1.2 Objectives . . . 16
1.3 Methodology . . . 17
1.4 Scope . . . 17
1.5 Dissemination Plan . . . 18
1.6 Document Outline . . . 19

2 Literature Review **21**

2.1 Modelling Material Behaviour . . . 21
2.1.1 Metallurgy and Observable Phenomena . . . 21
2.1.2 Stress, Strain, and Classical Plasticity . . . 22
2.1.3 Constitutive Equations . . . 24
2.1.4 Fitting Models to Data . . . 26
2.2 Mechanical Testing . . . 27
2.2.1 Mechanical Testing Apparatus . . . 27
2.2.2 Uniaxial Tensile Tests . . . 28
2.2.3 Plane-Strain Tension Tests . . . 30
2.2.4 Uniaxial Compression Tests . . . 31
2.2.5 Plane-Strain Compression Tests . . . 32

2.2.6 Temperature and Strain-Rate Considerations . . . 33
2.2.7 Documenting Test Results . . . 35

3 Software Design 38

3.1 Work Item Breakdown . . . 38
3.2 Dataset Processing Strategy . . . 41
3.3 List of Requirements . . . 46

4 Implementation and Usage 49

4.1 Paramaterial . . . 49
4.1.1 Overview of Paramaterial . . . 49
4.1.2 User Interface . . . 51
4.1.3 Installation and Dependencies . . . 51
4.1.4 Contributing or Logging Issues . . . 53
4.2 Usage Example . . . 53
4.2.1 Data Preparation Example . . . 53
4.2.2 Data Processing . . . 59
4.2.3 Data Aggregation . . . 64
4.2.4 Quality Control . . . 69
4.3 Important Algorithms . . . 75
4.3.1 Processing Algorithms . . . 75
4.3.2 Aggregation Algorithms . . . 76
4.3.3 Model-Fitting Algorithms . . . 78

5 Demonstration Case Studies 81

5.1 Variability in Material Properties of AA6061 . . . 82
5.2 Comparing Plane-Strain and Uniaxial Tension . . . 87
5.3 Effect of Heat Treatment on As-Cast AA3104 . . . 92
5.4 Plane-Strain Compression Geometry Effects . . . 97

6 Discussion 102

6.1 Discussion of Functionalities . . . 102
6.2 Discussion of Datasets . . . 106

7 Conclusions 112

7.1 Summary . . . 112
7.2 Key Outcomes . . . 114

7.3 Recommendations 115
7.4 Reflecting on Objectives 116
7.5 Closing Remarks 116

References **121**

A Additional Mechanical Testing Derivations **122**
A.1 Ideal Plane-Strain Conditions 122
A.2 Friction Correction for Uniaxial Compression Tests 124
A.3 Corrections for Plane-Strain Compression Tests 125
A.3.1 Breadth-Spread Correction 125
A.3.2 Friction Correction 126
A.3.3 Isothermal Correction 127

B Additional Implementation Details **128**
B.1 Architecture of Paramaterial 128
B.2 Contents and Structure of the Repository 129
B.3 Important Classes 131
B.4 Important Functions 133
B.5 Clickable GUI for Selecting Points 135
B.6 Branding 136
B.7 Paramaterial Usage Example Notebook 138
B.7.1 Processing 140
B.7.2 Data Aggregation 142
B.7.3 Quality Control 144

C Processing Notebooks **146**
C.1 CS1 Processing Report 146
C.2 CS2 Processing Report 152
C.3 CS3 Processing Report 157
C.4 CS4 Processing Report 164

List of Figures

2.1 Specimen clamped in universal testing machine. 27
2.2 Schematic of a tensile test specimen. 28
2.3 Typical engineering stress-strain curve. 29
2.4 Schematic of a plane-strain tension test specimen. 30
2.5 Schematic of a uniaxial compression test specimen. 31
2.6 Schematic of a plane-strain compression test specimen. 32

3.1 Venn diagram for work to process mechanical test datasets. 38
3.2 Phases of the dataset processing strategy. 41
3.3 Data storage file structure diagram. 43
3.4 Flow-chart showing the different phases of the use-case strategy. 45

4.1 Paramaterial logo. 49
4.2 Using Paramaterial to process large stress-strain datasets. 50
4.3 Using Paramaterial to make a table of parameters for analysis. 50
4.4 Experimental matrix showing distribution of tests across temperature and lot. . . 56
4.5 Stress-strain curves from the prepared uniaxial tension test data. 57
4.6 Stress-strain curves for the prepared example data as subplots. 58
4.7 Trimmed example data. 60
4.8 Example data after foot correction. 60
4.9 Screenshot from the screening PDF used in the example. 62
4.10 Screened and foot-corrected example data. 63
4.11 Processed example data plotted on a grid of subplots. 65
4.12 Representative curves for the processed example data. 66
4.13 Ramberg-Osgood model curves fitted to the example data. 68
4.14 Comparison of identified mechanical properties against literature. 70
4.15 Grid of subplots used for identifying outliers in the example data. 72

4.16 Screenshot from the tests receipts PDF. 74

5.1 Experimental matrix for CS1. 82
5.2 Mechanical properties for CS1 plotted against temperature. 83
5.3 Raw data, trimmed data, representative curves, and fitted models for CS1 84
5.4 Ramberg-Osgood model hardening coefficient values for CS1. 86
5.5 Experimental matrix for CS2. 87
5.6 Stress-strain curves of unprocessed CS2 example data. 87
5.7 Processed, fitted, and representative curves for CS2 88
5.8 Hardening coefficient values for the plane-strain and uniaxial test types. 89
5.9 Mechanical properties for UT and PST data. 90
5.10 Experimental matrix for CS3. 92
5.11 Flow stress against temperature for AA3104 in the AC, H560, and H580 states. . 92
5.12 Uniaxial compression test data for AA3104 at a strain-rate of $1\,s^{-1}$ and $10\,s^{-1}$. . . 93
5.13 Fitted Zener-Holloman model and conformance values for CS3 95
5.14 Experimental matrix for CS4. 97
5.15 Raw data and representative curves for PSC and PSC* tests. 97
5.16 Flow stress against temperature for AA3104 transfer bar 98
5.17 Fitted Zener-Holloman model and conformance values for CS4 99

B.1 File tree showing structure and contents of the Paramaterial project repository. . 129
B.2 UML class diagram for the data handling classes. 131
B.3 UML class diagram for the Styler class. 132
B.4 UML class diagram for the model fitting class. 132
B.5 UML class diagram for the test receipt generation class. 133
B.6 The GUI developed for manually identifying mechanical properties 135
B.7 Paramaterial logo. 136
B.8 Paramaterial icon design. 136

List of Tables

3.1 Descriptions of work items for processing a mechanical tests dataset. 39
3.2 Examples of metadata features for a processed mechanical test dataset. 43
3.3 Descriptions of inputs and outputs for each phase. 45

4.1 Package descriptions for the Paramaterial dependencies. 52
4.2 Metadata for prepared dataset. 55
4.3 Tests that were rejected during screening. 63
4.4 Representative info table. 67
4.5 Fitted parameters table. 68
4.6 Mechanical properties for the example dataset obtained using Paramaterial. . . . 69
4.7 Mechanical properties for the example dataset reported in the literature. 69

5.1 Mechanical properties determined for the CS1 example dataset. 82
5.2 Model parameters for CS1. 85
5.3 Model parameters for CS2 88
5.4 Mechanical properties for CS2. 89
5.5 Flow-stress and Zener-Holloman values for CS3. 94
5.6 Flow stress and Zener-Holloman values for CS4. 98

6.1 Terms of reference for data quality comparison. 107
6.2 Quality scores for the example datasets. 110

List of Listings

4.1 Run this Python script to automatically download the example. 54
4.2 Import the Paramaterial package and print out the version number. 54
4.3 Load the data and metadata into a DataSet object and check the formatting. . . . 54
4.4 Sorting the dataset and accessing the metadata. 54
4.5 Filtering the dataset to extract a subset. 55
4.6 Making an experimental matrix heatmap plot using Paramaterial. 55
4.7 Setting up a Styler object for plotting functions. 56
4.8 Defining a function that plots the entire dataset on a single plot. 56
4.9 Defining a function for plotting on a grid of subplots. 57
4.10 Automatically finding the ultimate tensile strength and fracture point. 59
4.11 Defining a trimming function and applying it to a dataset. 59
4.12 Identifying the proportional limits and applying foot correction. 60
4.13 Generating a screening PDF. 61
4.14 Reading the entries from a screening PDF into a DataSet object. 62
4.15 Rejecting tests that were flagged during screening. 63
4.16 Determining the proof stresses for a dataset of stress-strain curves. 64
4.17 Saving the processed data and visualising it on a grid of subplots. 64
4.18 Making representative data according to temperature groupings. 66
4.19 Fitting a material model to a dataset using Paramaterial. 67
4.20 Generating a dataset of predicted data using a fitted modelset. 68
4.21 Making a grid of subplots with representative curves to check for outliers. 71
4.22 Setting up the TestReceipts class and parsing the placeholders from a template. . 73
4.23 Generating a PDF of test reports for a dataset. 73

List of Algorithms

1 Finding the proportional limits of a stress-strain curve using linear regression. . 75
2 Applying foot correction. 76
3 Finding proof stress. 76
4 Finding combinations of key-value pairs for given categories to use as filters. . . 77
5 Creating representative data. 77
6 Return-mapping algorithm. 78
7 Fitting a constitutive equation to stress-strain data. 79

Acronyms

AC As-cast

ARG Aluminium Research Group

CERECAM Centre for Research in Computational and Applied Mechanics

CME Centre for Materials Engineering

GUI Graphical user interface

H560 Homogenised at 560 °C.

H580 Homogenised at 580 °C.

LPL Lower proportional limit

PS Proof strength

PSC Plane-strain compression

PST Plane-strain tension

PyPI Python Package Index

RMSE Root Mean Squared Error

TB Transfer bar.

UC Uniaxial compression

UCT University of Cape Town

UPL Upper proportional limit

UT Uniaxial tension

UTS Ultimate tensile strength

1 Introduction

> In the beginning there was nothing,
> which exploded.
>
> *Terry Pratchett*

THE project background is given in Section 1.1, before the objectives are presented in Section 1.2. The methodology is described in Section 1.3, and the scope is discussed in Section 1.4. Finally, the dissemination plan is specified in Section 1.5, and the document outline is presented in Section 1.6.

1.1 Background

The behaviour and properties of materials can be characterised using results from mechanical tests. This typically involves applying a prescribed deformation to a specimen and measuring the load required to do so[1]. However, the raw experimental data must be processed prior to analysis.

Processing mechanical test data involves making the data more suitable for analysis, by applying transformations to datasets consisting of time-series measurements and associated test information. A toolkit for automating the various processing steps would help to ensure the repeatability and quality of subsequent analysis. The development of such a toolkit as an open-source Python package is the focus of this project.

Testing the mechanical properties and behaviour of engineering materials is important for predicting failure of structural components, for developing accurate models, for controlling industrial processes, and for materials design [1, 2, 3, 4]. Mechanical testing procedures are well established, and are covered in multiple textbooks [1, 3, 5, 6, 7]. Standards organisations, for example ASTM and ISO, have published guidelines for obtaining valid and comparable test results. Several international companies[2] manufacture and sell mechanical testing equipment.

The Centre for Materials Engineering (CME) at the University of Cape Town (UCT) facilitates mechanical testing and associated studies. Researchers from CME and from the Centre for Research in Computational and Applied Mechanics (CERECAM), have formed the Aluminium

[1] Alternatively, the applied load can be specified, and the resulting deformation measured.
[2] See Instron, Zwick Roell, and Gleeble.

Research Group (ARG). The cross-discipline group utilises mechanical testing, microstructural studies, computational simulations, and data analytics to characterise and model the properties and behaviour of aluminium alloys during thermo-mechanical processing. This research is bolstered by working closely with Hulamin, an industrial semi-fabricator of aluminium products.

The coupling of physical experiments with computational modelling necessitates well-managed data communication between experimentalists and analysts [8], since large datasets of high-quality mechanical test results are required for accurate computational modelling and data analysis [9, 10]. However, raw measurements must be cleaned and post-processed. Doing so manually is unmanageably laborious, and introduces human error. Hence, processing of experimental data should be automated where possible.

It is therefore likely that a toolkit for processing the mechanical test data produced by experimentalists would help improve the quality and quantity of data provided to analysts. Existing solutions are either proprietary[1], or are designed for use by individual research groups [11]. The initial development of an open-source solution is presented in this dissertation.

1.2 Objectives

The aim of this project is to develop a software toolkit for repeatable processing of experimental results from mechanical testing of materials. The toolkit should help to ensure that the processing and analysis of material test datasets is repeatable and traceable. This would help improve the credibility of conclusions from subsequent analysis of processed datasets, and make it simpler for future studies to build on previous work.

Additionally, the toolkit should help to enhance the quality and quantity of test data by providing functionality for documentation of processing procedures and increasing efficiency in processing large datasets. The following primary objectives have been identified to achieve this aim:

Obj. 1. *Develop an extensible software toolkit for processing mechanical test data.*

Obj. 2. *Demonstrate the toolkit functionalities by processing existing mechanical testing datasets.*

Obj. 3. *Document and deploy the toolkit so that other researchers can utilise existing functionalities, or extend the software capabilities.*

[1] See Origin and Granta.

1.3 Methodology

The project methodology is summarised as 'Research, Design, Implement, Demonstrate, Deploy'. The phases of this methodology are described as follows:

1. **Research**: The research phase involves conducting a review of relevant literature in preparation for defining the software requirements. This includes investigating industrial and academic use-cases for mechanical test data, as well as studying experimental guidelines and standards for relevant mechanical tests.

2. **Design**: The design phase involves outlining the software requirements. These are used to identify specific functionalities for implementation, and inform the overall software architecture.

3. **Implement**: The implementation phase involves writing the software and leveraging open-source data science libraries to implement the functionalities identified in the design phase.

4. **Demonstrate**: The demonstration phase involves showcasing the effectiveness of the software through a series of case studies using example datasets for aluminium. These were gathered from previous students and open-source databases.

5. **Deploy**: The deployment phase entails packaging the software and making it available for download and installation. To this end, detailed code documentation is hosted on GitHub alongside the source code.

1.4 Scope

The primary objective of the project is to develop a software toolkit that can effectively process data, making it ready for further analysis. The focus of the project is not on making specific research findings through data analysis, but rather on providing a tool that can help users prepare their data for analysis. The experimental procedures used to generate data are also not the focus, although some commentary on experimental procedures is provided.

Hence, the project outcomes and limitations are as follows:

- *Project Outcomes:*
  - A strategy comprised of four phases – preparation, processing (including cleaning and identifying mechanical properties), aggregation (including modelling) and quality control – has been established for the processing of mechanical test datasets.
  - An open-source Python toolkit was developed to automate mechanical test data

processing and analysis. The toolkit provides a streamlined workflow, minimizes errors, and optimises efficiency for users in various material science fields.

  - The capabilities of the toolkit are demonstrated using actual mechanical test datasets, showcasing its effectiveness.
  - The toolkit is accompanied by documentation, tutorials, and sample code designed to guide users in using its features and functionality. This promotes its adoption and facilitates the implementation of best practices in data processing and analysis.
  - The example datasets, which contain data from hundreds of tests that have been processed using Paramaterial, constitute a valuable database for future research in the field.

- *Project Limitations:*
  - No new experimental data has been produced in this project. However, a significant outcome of the project was the creation of a unified database from the collected example datasets. This compiled example dataset serves as a resource for further research.
  - A structured consultation process to obtain and incorporate software requirements from stakeholders was not undertaken. Instead, the requirements were established in an informal manner by evaluating functionalities while processing the collected example datasets and determining what was effective and what was not.
  - The mechanical tests demonstrated in the case studies are limited to uniaxial tension, plane-strain tension, uniaxial compression, and plane-strain compression tests[1].

The software toolkit developed in this project has the potential for improving the quality of mechanical test data in both academia and industry. Its impact depends on the extent of its adoption.

1.5 Dissemination Plan

The source code is published on GitHub, along with code documentation and usage tutorials. The software is also available as a Python package on PyPI. Furthermore, the raw and processed data and dataset reports will be made available on ZivaHub, the cloud storage platform prescribed by UCT. The relevant links are as follows:

Python Package: https://pypi.org/project/paramaterial/
Source Code: https://github.com/dan-slater/paramaterial
Code Documentation: https://dan-slater.github.io/paramaterial/
ZivaHub: https://zivahub.uct.ac.za/

[1] See Sections 2.2.2, 2.2.3, 2.2.4, and 2.2.5.

1.6 Document Outline

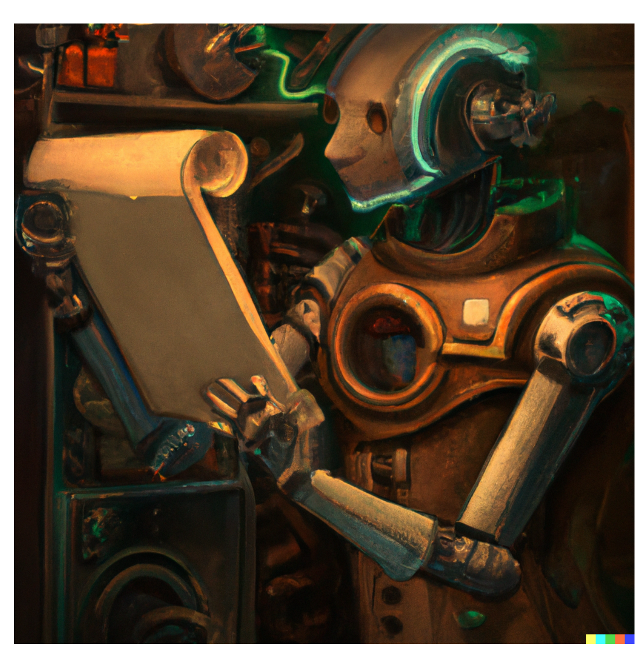
A literature review is presented in **Chapter 2**. The literature review covers the basics of modelling the behaviour of metallic materials, with a focus on plasticity. Mechanical testing procedures are also covered in this chapter, and specific details are given for the test types considered in this dissertation: uniaxial tension, uniaxial compression, plane-strain tension, and plane-strain compression.

A general strategy for repeatable processing of mechanical test datasets is then given in **Chapter 3**. Software design requirements are also presented in this chapter. The software package that was developed to meet these requirements is then presented in **Chapter 4**. Usage instructions are given, functionalities are outlined, and important algorithms are presented.

Next, the capabilities of the software are demonstrated through various case studies in **Chapter 5**. Four example datasets are processed and analysed using the software, highlighting its efficiency and adaptability to different datasets.

An analysis of the software's functionality in relation to the case studies is given in **Chapter 6**, along with critical commentary and comparison of the different example datasets. Finally, conclusions and recommendations for future work are provided in **Chapter 7**.

Appendix A contains additional mechanical testing derivations for several detailed data-correction procedures. Further implementation details are given in **Appendix B**, including the full Jupyter Notebook for the usage example presented in Chapter 4. The Jupyter Notebooks for the case studies are also included in **Appendix C**. The code in these Jupyter Notebooks can be run to obtain the results presented in the earlier chapters of this report, making the usage example and case studies fully repeatable.



2 Literature Review

> The more that you read, the more things you will know. The more that you learn, the more places you'll go.
>
> *Dr. Seuss*

THIS *literature review is divided into two sections. Section 2.1 contains a review of models for the material behaviour of metallic materials, and Section 2.2 contains a review of relevant mechanical testing techniques.*

2.1 Modelling Material Behaviour

The structure and resulting behaviour of metallic materials under loading is discussed in Section 2.1.1. Thereafter, the concepts of stress and strain are introduced in Section 2.1.2. A review of common material models is given in Section 2.1.3, and the procedure for fitting models to data is discussed in Section 2.1.4.

2.1.1 Metallurgy and Observable Phenomena

Structural components are typically at a scale of metres. However, understanding and predicting the performance of engineering materials requires knowledge of mechanisms that exist over the entire range of scales from the engineering scale, 10^0 m, down to the size of atoms, around 10^{-10} m [12]. At the nanometre scale, atoms are assembled into simple geometric cells, termed unit cells, that are repeated to form a crystal lattice structure.

If deformation is sufficiently small, that is, if the *elastic limit* is not reached, atomistic bonds remain intact and the structure returns to its initial state upon removal of the applied loading. Stretching of bonds thus allows solid metals to behave elastically. Deformations greater than the elastic limit induce breaking of bonds and reorganisation of the lattice. This leads to plasticity, that is, permanent deformation [13]. A permanently deformed metal is said to have yielded, and *yield strength* is a measure of the load that can be applied to a material before its elastic limit is exceeded.

For many metals, plastic deformation increases the apparent yield strength of the material. This phenomenon is known as *work-hardening* [14]. In contrast, an increase in temperature typically

leads to a decrease in yield strength, known as *softening* [15]. The deformation-response of metals also depends on the strain-rate. However, this rate-dependence is usually negligible at temperatures below $0.35T_m$, where T_m is the melting temperature in Kelvin [16].

Models for the behaviour of liquid or close-to-liquid metals therefore often do not depend on strain. Conversely, solid metals are often modelled as rate-independent.

2.1.2 Stress, Strain, and Classical Plasticity

Stress and strain are two important concepts in mechanics that describe the behaviour of a material under load. Stress is a measure of the internal forces within a material. Strain is a measure of the deformation of a material. Definitions for these measures in a three-dimensional context can be found in [18, 19, 20]. However, simple one-dimensional notions of stress and strain are sufficient for the work presented in this report.

Engineering stress, s, is defined as the force, F, per unit undeformed area, A_0,

$$s = \frac{F}{A_0}\,. \tag{2.1}$$

An alternative stress measure is the true stress, σ, defined as the force per unit deformed area, A,

$$\sigma = \frac{F}{A}\,. \tag{2.2}$$

Engineering, or nominal strain, e, is defined as the ratio of the change in length, Δl, to the undeformed length, l_0,

$$e = \frac{\Delta L}{L_0}\,. \tag{2.3}$$

The true, or logarithmic strain, ε, is defined in an incremental setting[1] as the ratio of an increment of length to the deformed length,

$$\mathrm{d}\varepsilon = \frac{\mathrm{d}L}{L}\,. \tag{2.4}$$

The instantaneous true strain is then

$$\varepsilon = \ln\frac{L}{L_0}\,. \tag{2.5}$$

For small-strain elasticity, stress is related to strain by Hooke's law,

$$\sigma = E\varepsilon\,, \tag{2.6}$$

where E is the Young's modulus, a material parameter[2].

[1] An increment is defined as $\mathrm{d}\bullet := \lim_{\Delta t\to 0} \bullet(t+\Delta t) - \bullet(t)$.

[2] A parameter that must be determined for a specific material in a given state.

If a material is deformed beyond its elastic limit, the resulting strain exceeds what is predicted by Hooke's law. If rate-sensitivity is neglected, this additional strain can be defined as the plastic strain [14], ε_p, given by

$$\varepsilon_p = \varepsilon - \frac{\sigma}{E}\,. \tag{2.7}$$

The initial loading path that causes plastic deformation is called the *virgin curve*. The elastic limit on the virgin curve is called the initial yield strength, σ_Y, which is a material parameter. If the material is unloaded and then reloaded, the instantaneous yield strength, or flow strength, denoted by σ_f, may have increased or decreased, depending on whether the plastic deformation resulted in hardening or softening. The following equations from one-dimensional classical plasticity theory can be used to model this elastic-plastic, loading-unloading behaviour [18], where γ is the plastic flow rate and α is the accumulated plastic strain[1]:

$$\text{Elastic stress-strain relation:} \quad \sigma = E(\varepsilon - \varepsilon_p)\,, \tag{2.8}$$

$$\text{Flow rule with isotropic hardening:} \quad \dot{\varepsilon}_p = \gamma\,\mathrm{sign}(\sigma)\,, \tag{2.9}$$

$$\dot{\alpha} = \gamma\,, \tag{2.10}$$

$$\text{Yield condition:} \quad f(\sigma,\alpha) = |\sigma| - \sigma_f(\alpha) \leq 0 \tag{2.11}$$

$$\text{Kuhn-Tucker complementarity conditions:} \quad f(\sigma,\alpha) \leq 0, \quad \gamma \geq 0, \quad \gamma f(\sigma,\alpha) = 0\,, \tag{2.12}$$

$$\text{Consistency condition:} \quad \gamma \dot{f}(\sigma,\alpha) = 0 \quad \text{if } f(\sigma,\alpha) = 0\,. \tag{2.13}$$

The stress-strain relation can then be extended to include plastic deformation by using a piecewise split,

$$\sigma = \begin{cases} E(\varepsilon - \varepsilon_p)\,, & \text{if } \sigma < \sigma_f(\alpha)\,, \\ \sigma_f(\alpha)\,, & \text{otherwise}\,, \end{cases} \tag{2.14}$$

where Hooke's law applies if the stress is lower than the flow strength of the material, which evolves as function of the accumulated plastic strain.

[1] Equations (2.9) and (2.10) can be restated as $\dot{\alpha} = |\dot{\varepsilon}_p|$.

2.1.3 Constitutive Equations

A material model, or *constitutive equation*, can be used to predict stress as a function of kinematic variables like strain and strain-rate, as well as state variables like temperature, plastic strain and accumulated plastic strain. Equation (2.14), for example, predicts stress as a function of strain during elastic deformation, and as a function of the accumulated plastic strain during plastic deformation. However, an expression for the flow strength is still required. The following models can be used for this purpose:

$$\text{Ramberg-Osgood [21]:} \quad \sigma_f = \sigma_Y + K\alpha^n \tag{2.15}$$

$$\text{Voce [22]:} \quad \sigma_f = \sigma_s + (\sigma_Y - \sigma_s)\left(1 - e^{\delta\alpha}\right), \tag{2.16}$$

$$\text{Modified Ludwik [23]:} \quad \sigma_f = \sigma_Y + K(\alpha)^n(\dot{\alpha}^*)^m, \tag{2.17}$$

$$\text{Johnson-Cook [24]:} \quad \sigma_f = [\sigma_Y + K\alpha^n][1 + C\ln(\dot{\alpha}^*)][1 - (T^*)^g], \tag{2.18}$$

$$\text{Zerilli-Armstrong [25]:} \quad \sigma_f = \sigma_Y + K\sqrt{\alpha}\exp(-D_0T + D_1T\ln\dot{\alpha}^*). \tag{2.19}$$

In equations (2.15) to (2.19), α is the accumulated plastic strain, $\dot{\alpha}^*$ is the dimensionless rate of plastic strain accumulation[1], K is a hardening modulus in MPa, n is the dimensionless strain-hardening exponent, m is the dimensionless strain-rate sensitivity exponent, and σ_Y is the initial yield strength in MPa. The Ramberg-Osgood model has been shown to be suitable for modelling aluminium alloys and other metals in both tension and compression [26].

In the Voce model, σ_s is the saturation stress in MPa, that is, the stress at which a material reaches its maximum strength, and δ is a dimensionless material parameter. The Ramberg-Osgood, Voce, and Modified Ludwik are used for materials that exhibit hardening behaviour.

The Johnson-Cook model, in which C and g are dimensionless parameters and $T^* = T/T_m$ is the homologous temperature[2], can describe hardening behaviour, as well as thermal softening effects. This is also the case for the Zerilli-Armstrong model, in which D_0 and D_1 are dimensionless material parameters.

Models that do not describe phenomena like thermal softening can be modified to do so. The reader is referred to [27] for an example of such a modification, and to [26] for a history of the above and of similar empirical models.

For the constitutive equations that follow, it is useful to define the characteristic flow stress, denoted by σ_F. The characteristic flow stress of a particular load path is the flow stress at a specified strain value. Alternatively, the characteristic flow stress can be obtained as the average flow stress over

[1] Given by $\dot{\alpha}^* = |\dot{\varepsilon}_p|/\dot{\varepsilon}_0$, where $\dot{\varepsilon}_0$ is a reference rate.

[2] The homologous temperature, also known as the reduced temperature, is a dimensionless quantity used to describe the thermal state of a material. T_m is the melting point temperature.

a range of strains, or as the maximum flow stress over a range of strains. It is important to note that σ_F refers to a single value per curve, while σ_f represents the entire curve. Material models can be used to relate material parameters like the characteristic flow stress or yield strength to the temperature and strain-rate, without considering strain as an independent variable.

A common approach is to use the Zener-Holloman [28] parameter, which describes the strain-rate and temperature sensitivity of a material.

$$\text{Zener-Holloman Parameter:} \qquad Z = \dot{\varepsilon} \exp\left(\frac{Q}{RT}\right) \tag{2.20}$$

$$\text{Strain-Rate Relation:} \qquad \dot{\varepsilon} = A \exp(B\sigma_F) \exp\left(-\frac{Q}{RT}\right) \tag{2.21}$$

$$\text{Flow-Stress Relation:} \qquad \sigma_F = \frac{1}{B} \ln Z - \frac{1}{B} \ln A \tag{2.22}$$

Here, Q is the activation energy in kJ mol^{-1}, R is the universal gas constant in J K^{-1} mol^{-1}, T is the temperature in K, and A and B are dimensionless material parameters.

Whereas equation (2.22) is only suitable for high stresses, the Arrhenius-type model, developed in [29, 30, 31, 32], can be used for both low and high stresses. The exponential in equation (2.21) is replaced by a hyperbolic power law so that the strain-rate relation becomes

$$\dot{\varepsilon} = A[\sinh(B\sigma_F)]^{n_z} \exp\left(-\frac{Q}{RT}\right), \tag{2.23}$$

with A, B and n_z as material parameters. The flow-stress relation can then be expressed as

$$\sigma_F = \frac{1}{B} \ln\left\{\left(\frac{Z}{A}\right)^{1/n_z} + \left[\left(\frac{Z}{A}\right)^{2/n_z} + 1\right]^{1/2}\right\}. \tag{2.24}$$

Material models containing the Zener-Holloman parameter are commonly used for simulating hot forming operations [33]. However, it is noted that Q cannot always be treated as a constant over different strain-rates and temperatures [34]. A remedy is presented in [35], where the activation energy is given by

$$Q = R\left[\frac{\partial \ln(\dot{\varepsilon})}{\partial \ln(\sigma_F)}\right]_{T=const}\left[\frac{\partial \ln(\sigma_F)}{\partial \ln(1/T)}\right]_{\dot{\varepsilon}=const}. \tag{2.25}$$

2.1.4 Fitting Models to Data

Before a model can be used to predict the behaviour of a specific material, it must first be calibrated to match experimental data from that material. This is termed "fitting the model to the data", and involves finding values for the material parameters such that the model fits the data to within a specified error margin.

The Root Mean Squared Error, or RMSE, is a metric that can be used to quantify how similar the predicted and measured curves are for a given set of parameters. The RMSE is given by:

$$RMSE = \sqrt{\frac{\sum_i^N \left[\sigma(\varepsilon_i^*) - \sigma_i^*\right]^2}{N}}, \tag{2.26}$$

where ε_i^* and σ_i^* are pairs of measured stress-strain data points, $\sigma(\varepsilon_i^*)$ is the stress predicted by the constitutive equation for a given ε_i^*, and N is the number of data points used during fitting. The RMSE provides a measure of the average deviation of the predicted stress from the measured stress.

Another metric is the coefficient of determination, also known as R-squared (R^2). R^2 measures the proportion of the total variation in the dependent variable (stress, in this case) that can be explained by the independent variable (strain). R^2 is calculated as

$$R^2 = 1 - \frac{\sum_i^N \left[\sigma_i^* - \sigma(\varepsilon_i^*)\right]^2}{\sum_i^N \left[\sigma_i^* - \bar{\sigma}^*\right]^2}, \tag{2.27}$$

where $\bar{\sigma}^*$ is the mean of the measured stress values. R^2 values range between 0 and 1, with a value of 1 indicating a perfect fit, and 0 suggesting that the model does not explain any of the observed variation.

It is possible to determine which material model is best suited to a specific material by comparing the fitting errors. An example of this is given in [36], where the authors compared the Johnson-Cook, Zerilli-Armstrong, and Arrhenius-type models (equations (2.18), (2.19) and (2.24)) to investigate which of these would most accurately predict the stress-response of an SnSbCu alloy. Another approach is to combine several models to achieve better agreement with experimental data. The authors of [37], for example, obtained a better fit by using a combined Johnson-Cook and Zerilli-Armstrong model than what was possible with either of the models on their own.

An overview of modelling material behaviour has been provided in this section, covering concepts such as metallurgy, stress, strain, and classical plasticity. Various constitutive equations appropriate to metallic material behaviours were introduced, and the process of fitting models to experimental data was discussed. The focus of the next section is on the procedures used to generate experimental data.

2.2 Mechanical Testing

A general overview of mechanical testing apparatus is given in Section 2.2.1, followed by more detailed treatments of uniaxial tensile testing in Section 2.2.2, plane-strain tension testing in Section 2.2.3, uniaxial compression testing in Section 2.2.4, and plane-strain compression testing in Section 2.2.5.

2.2.1 Mechanical Testing Apparatus

Determining the mechanical response of materials to various loading scenarios is important for accurate simulation of manufacturing processes, for predicting failure of structural components, and for materials selection and design. Mechanical tests are used to extract specific properties from a material sample, or test specimen.

Mechanical tests on metallic materials are commonly performed on a universal testing machine (UTM), which can be used for a range of tests with applied tensile or compressive forces. UTMs can be electromechanical, which are driven by electrical motors, or servo-hydraulic. The latter are usually larger, and their hydraulic drive systems are able to operate at higher applied forces [38].

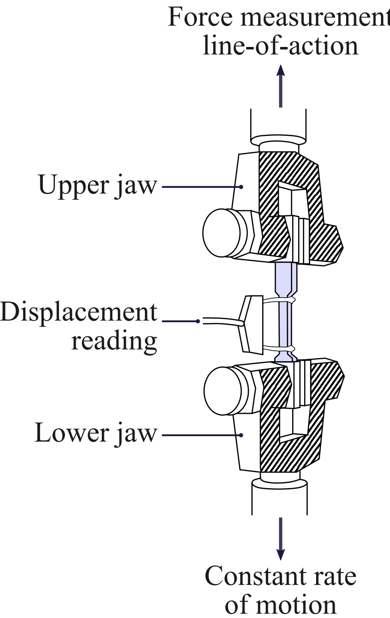


Figure 2.1: *A tensile test specimen clamped between two grips of a universal testing machine.*

To conduct tension experiments on a UTM, specimens are clamped between two grips, before the drive system is used to apply a tensile force. The grips can be swapped out for platens, between which compression test specimens are placed, before a compressive force is applied by the drive system. In both cases, the magnitude of the applied force is measured using a load-cell connected in series with the drive train; that is, along the force measurement line of action displayed in Figure 2.1.

The relative movement of the grips/platens can be measured using a linear-variable transducer, to give a reading for displacement. However, the transducer measurements should be corrected for compliance, so that the displacement accurately represents the deformation of the specimen itself. Compliance is the extension or compression of components or lubricant attached to the specimen between the transducer connection points, which causes an apparent additional displacement in the measured results. To perform the correction, a compliance test must be performed prior to the experiment, under the same conditions, but the specimen is replaced by a dummy specimen with comparatively high stiffness [39]. The displacement measured during the compliance test can then be subtracted from the actual test measurements, so that

$$d_{corrected} = d_{raw} - d_{compliance} \, , \tag{2.28}$$

where $d_{corrected}$ is the compliance-corrected displacement data, d_{raw} is the raw displacement data in need of correction, and $d_{compliance}$ is the displacement data from the compliance test. To avoid compliance-distorted readings, the displacement can be measured using an extensometer, which tracks two points marked on the specimen using laser or video.

2.2.2 Uniaxial Tensile Tests

Uniaxial tension (UT) tests are the most commonly used type of mechanical test, largely due to the simple experimental setup [14]. By applying a force, F, to a simple rod or bar-shaped specimen, a uniaxial stress-state can be induced in the material during deformation. The specimen is clamped on either side, as in Figure 2.1, and stretched along its axis until fracture. The larger ends, annotated in Figure 2.2, facilitate gripping, and ensure that plastic deformation occurs only in the gauge section, also annotated. Extension of the gauge section, ΔL, is given by the displacement measuring instrument.

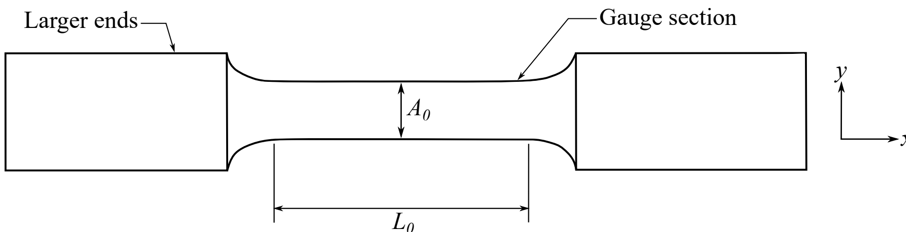


Figure 2.2: *Schematic of a tensile test specimen.*

The initial gauge length and cross-sectional area, L_0 and A_0, are used to convert the displacement and force measurements to engineering strain,

$$e = \frac{\Delta L}{L_0} \, , \tag{2.29}$$

and engineering stress,

$$s = \frac{F}{A_0}\,, \tag{2.30}$$

which can be plotted on the engineering stress-strain curve, see Figure 2.3. Various mechanical properties can then be graphically or algorithmically determined, including Young's modulus, yield strength, tensile strength and facture strength. For materials that do not have a defined yield strength, the 0.2% offset yield strength, or "proof strength" is often quoted. The proof strength is determined by the intercept of a line parallel to the linear portion of the curve with an x-intercept at 0.2% engineering strain.

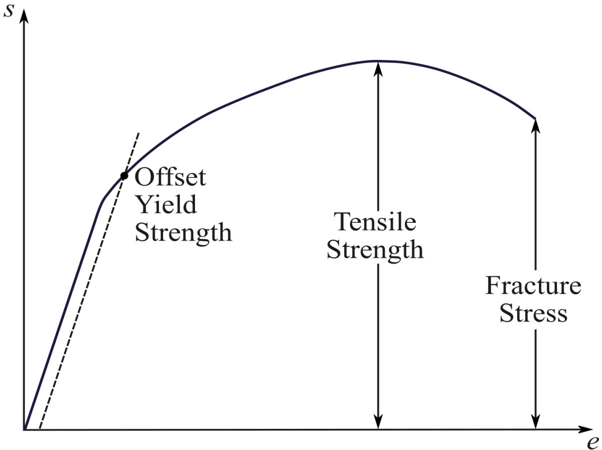


Figure 2.3: *An engineering stress-strain curve for a typical uniaxial tensile test on a material without a clearly defined yield point.*

Mechanical properties read off the engineering stress-strain curve can be used to compare materials. However, they do not accurately represent the mechanical behaviour, because of the false assumption that the gauge width and length remain constant. True stress and strain provide a better description, for which the current specimen geometry is taken into account. Plastic deformation is assumed to be isochoric and elastic deformation is comparatively small. Therefore, the volume remains approximately constant, so that the current length, L, and current area, A, can be related to the initial length and area using

$$A_0 L_0 = AL\,. \tag{2.31}$$

Equation (2.31) can be substituted into equation (2.2) to relate the engineering stress to the true stress for the uniaxial test,

$$\sigma = \frac{F}{A_0}\frac{A_0}{A} = s\left(\frac{L}{L_0}\right) = s\left(\frac{L_0 + \Delta L}{L_0}\right) = s(1+e)\,. \tag{2.32}$$

True strain can be related to its engineering analogue by using the definition of true strain,

$$\varepsilon = \int_{L_0}^{L} \frac{dl}{l} = \ln\left(\frac{L}{L_0}\right) = \ln\left(\frac{L_0 + \Delta L}{L_0}\right) = \ln(1+e) \tag{2.33}$$

The above treatment assumes the gauge section deforms uniformly [40]. However, in ductile materials, small geometric imperfections create a multi-axial stress concentration. This results in a sharp decrease in the cross-sectional area at the site of the imperfection, known as necking. For ductile metals, fracture of the specimen only occurs after substantial necking, making it difficult to predict [41]. Nonetheless, studying the behaviour of a tensile specimen after necking can still be useful, because the phenomenon of localisation also occurs in industrial forming operations [42].

2.2.3 Plane-Strain Tension Tests

Plane-strain tension tests are used to simulate metalworking processes that exhibit plane-strain conditions [5]. As with uniaxial tensile tests, the specimens are clamped on either side and stretched along a single axis, the x-axis in Figure 2.6. Due to the width of the specimen, strain is restricted

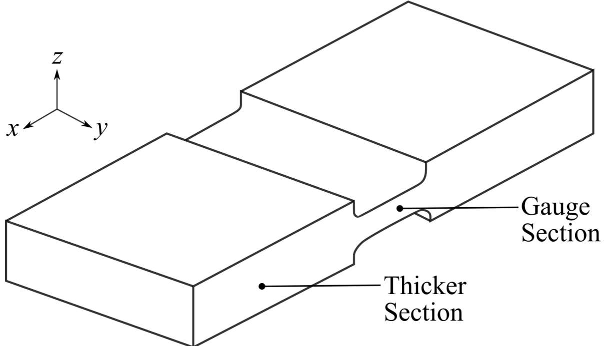


Figure 2.4: *Schematic of a plane-strain tension test specimen.*

to the x-z plane and $\varepsilon_y \approx 0$. In the gauge section, the specimen is thin in the z-direction, so that $\sigma_z \approx 0$. Hence, the gauge section is in a state of plane-strain. A force applied in the x-direction will generate an axial stress[1]

$$\sigma_x = \frac{F}{A_0}\left(\frac{L_0 + \Delta L}{L_0}\right), \tag{2.34}$$

and an axial strain

$$\varepsilon_x = \ln\left(\frac{L_0 + \Delta L}{L_0}\right), \tag{2.35}$$

where F is the applied force, A_0 is the initial cross-sectional area of the gauge section, and L_0 and ΔL are the initial length and extension of the gauge section.

If a stress-strain curve is plotted from the force and displacement measurements of a plane-strain tension test, the yield strength will appear to be higher than that measured using a uniaxial tension test. The measured stress and strain from a plane-strain tension test, σ_x and ε_x, can be related to an

[1] The axial stress, σ_x is calculated in the same was as for the tensile test in equation (2.32), but the gauge section of the plane-strain specimen differs in that there is also a non-zero secondary stress, σ_y. Similarly for the axial strain, ε_x, and secondary strain, ε_z.

equivalent tensile stress and strain by

$$\bar{\sigma} = \frac{\sqrt{3}}{2}\sigma_x \,, \tag{2.36}$$

$$\bar{\varepsilon} = \frac{2}{\sqrt{3}}\varepsilon_x \,. \tag{2.37}$$

Detailed derivations for equations (2.36) and (2.37) are given in Appendix A.1.

2.2.4 Uniaxial Compression Tests

The objective of uniaxial compression tests is to induce a uniform uniaxial stress state in a material sample. This is achieved by compressing a cylindrical sample between two platens, see Figure 2.5.

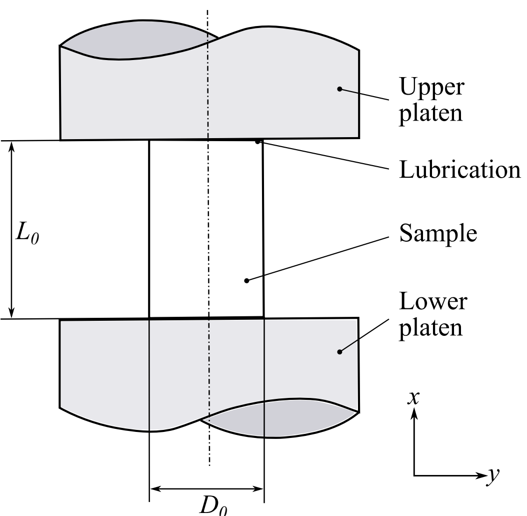


Figure 2.5: *Schematic of a uniaxial compression test specimen.*

When dealing with results from compression tests, it is convenient to treat force and displacement as positive in the compressive directions. Then, using the same approach as in equations (2.30) and (2.29), true stress in compression is given by

$$\sigma = \frac{F}{A_0}\left(\frac{L_0 - \Delta L}{L_0}\right) = s(1-e)\,, \tag{2.38}$$

and true strain in compression is given by,

$$\varepsilon = \ln\left(\frac{L_0 - \Delta L}{L_0}\right) = \ln(1-e)\,. \tag{2.39}$$

Equations (2.38) and (2.39) assume homogeneous deformation. However, contact between the sample and platens results in undesirable frictional forces that induce a non-uniform stress-state

in the material. The frictional forces restrict horizontal expansion at the interface, so that the deformed sample is wider at its centre than at its ends. This phenomenon is known as *barreling*. The platen-sample interface is usually lubricated to minimise this effect, but visible barrelling indicates that the results require a friction correction. Equations that can be used to perform such a correction are given in Appendix A.2.

Measurements of the initial and final geometry of the specimen can also be used to evaluate the extent of inhomogeneous deformation. Examples of test-validity metrics for uniaxial compression tests are the barrelling coefficient [43],

$$B_v = \frac{L_f D_f^2}{L_0 D_0^2}, \tag{2.40}$$

and the ovality coefficient [43],

$$O_v = \frac{D_{f\,max}}{D_{f\,min}}, \tag{2.41}$$

where L_f, L_0, D_f, and D_0 are the final and initial lengths and diameters of the specimen, and $D_{f\,max}$ and $D_{f\,min}$ are the largest and smallest of multiple diameter measurements taken at different angles on the final specimen.

2.2.5 Plane-Strain Compression Tests

Plane-strain compression (PSC) tests are used for identifying mechanical properties and developing constitutive equations appropriate for hot rolling models. They are preferable over other tests for this application because the strain and temperature distributions in the specimens during deformation resemble those encountered during rolling [44]. Under ideal plane-strain conditions, strain is

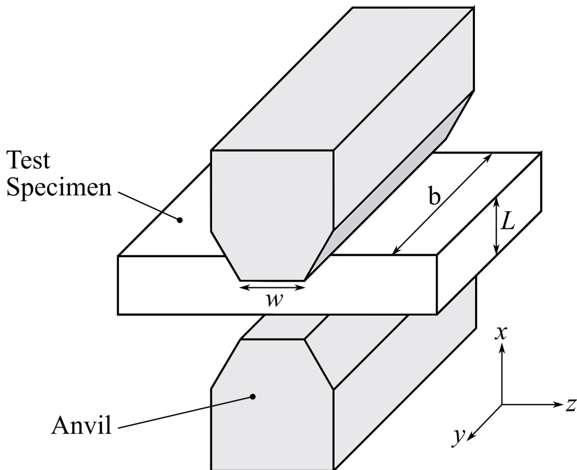


Figure 2.6: *Schematic of a plane-strain compression test specimen.*

restricted to the x-z plane (see Figure 2.6), and $\varepsilon_y \approx 0$. If friction is neglected, nothing prevents

deformation in the z-direction, so that σ_z is zero [45]. The stress and strain along the x-axis are given by

$$\sigma_x = \frac{F}{wb}\left(\frac{L_0 - \Delta L}{L_0}\right) , \tag{2.42}$$

$$\varepsilon_x = \ln\left(\frac{L_0 - \Delta L}{L_0}\right) . \tag{2.43}$$

The equivalent stress and equivalent strain can be then calculated using equations (2.37) and (2.36), as for the plane-strain tension test.

It has been shown that determining true stress-strain data from the raw load displacement data requires corrections for zero offset, machine compliance, extrusion of lubricant, increase in testpiece breadth, and friction [46]. Detailed derivations for these corrections are given in Appendix A, but the resulting equations are complex. As a simpler alternative, the following empirically based equations have been shown to produce acceptable results [47]:

$$\bar{\sigma} = B\frac{F}{wb} , \tag{2.44}$$

$$\bar{\varepsilon} = \frac{1}{A}\ln\left(\frac{L_0 - \Delta L}{L_0}\right) , \tag{2.45}$$

where the constants A and B must be determined for different specimen geometries and material-lubricant combinations [45].

Temperature gradients caused by deformational heating are of particular concern in PSC tests [48], and the assumption of isothermal conditions has been shown to produce inaccurate results [49]. Therefore, when using the measured stress-strain curves to derive constitutive equations it is important to use the instantaneous value for temperature at each point [32]. The constitutive equation can then be used to interpolate between temperature values[1] for each of the measured stress-strain points, so that an equivalent isothermal curve can be derived for comparison against results from other tests [50].

2.2.6 Temperature and Strain-Rate Considerations

Material tests are designed to reduce the complexity of the applied deformation and the associated material response, so that specific mechanical properties can be measured. Tension and compression tests are therefore usually conducted at nominally constant temperatures and strain-rates, and multiple tests are performed to determine the temperature and strain-rate dependency of mechanical properties.

To test for mechanical properties at elevated temperatures, a mechanism for heating and maintaining

[1]See Appendix A.3.3 for isothermal correction equations.

the temperature of the specimen is required. A furnace-like chamber can be used to heat the enclosed atmosphere and materials, or resistive heating can be used to heat the specimen directly. Temperature measurements are then obtained via thermocouples attached to the specimen. The heating path data from measurements taken prior to deformation should be included in the test report [51], because different thermal histories affect the mechanical properties of the material [52]. Additionally, thermal expansion needs to be considered when measuring initial and final specimen geometries at room temperature [46]. The change in length of a specimen due to thermal expansion, ΔL_T, as a function of the temperature change, is given by

$$\Delta L_T = L_f - L_0 = L_0\,\alpha_T \Delta T. \tag{2.46}$$

Here, α_T is the coefficient of thermal expansion, which can also be used to calculate heat of deformation. Assuming adiabatic conditions, the mean temperature rise is predicted by

$$\Delta T = \int_0^{\bar{\varepsilon}} \frac{p}{\alpha_T \rho}\, d\varepsilon\,, \tag{2.47}$$

where ρ is density. The average pressure, p, and equivalent strain, $\bar{\varepsilon}$, can be calculated from the load-displacement curve [50]. To perform origin correction the initial and final measured lengths, L_0 and L_f, are first corrected for thermal expansion [46]. The actual change in length of the specimen during hot deformation is then

$$\Delta L_{specimen} = L_f - L_0\,. \tag{2.48}$$

The displacement data, ΔL, can then be adjusted to match, so that

$$\Delta L_{corrected} = \Delta L - \max(\Delta L) + \Delta L_{specimen}\,. \tag{2.49}$$

UTMs can be used for tests at rates of up to 10^2 s^{-1}, but electro-mechanical UTMs are more suitable for lower rates, while servo-hydraulic UTMs are more suitable for higher rates. If a testing apparatus does not measure velocity directly, the instantaneous strain-rate can be calculated from the derivative of a polynomial fitted to the strain-time curve [45].

The phenomenon of *load-cell ringing*, that is, vibration of the load-cell, can also distort the force signal at high strain rates. To achieve high strain rates that are constant throughout the applied deformation, part of the load-train needs to be accelerated and impacted. This causes components to resonate at their natural frequencies [51]. The resulting vibration confounds the measurements obtained by transducers. The frequency of oscillations is dependent on the stiffness and mass of the load train. This effect can be minimised by using high-quality sensors, or by adjusting the setup to use a lower impact velocity [53].

2.2.7 Documenting Test Results

Standards and best practice documents have been developed to allow for comparison between tests from different laboratories and to minimise uncertainty in measurements. These documents provide guidelines for which information should be recorded in a report for each individual test. The process of analysing and correcting test measurements can be involved, but each post-processing step should be recorded and documented in the test report. An extensive, but not exhaustive, list of information that should be included in test reports is as follows:

- From the ASTM tensile testing standard [54]:
  - A reference to the standard used.
  - Information to identify the material and sample, for example the lot number of a sheet, and the location and orientation from which the sample was cut.
  - Specimen type, which defines initial specimen geometry.
  - The value of and method for determining the yield strength.
  - Original gauge length, percentage increase, and method used to determine elongation.
  - Procedures used for any analysis included in the report.

- From the measurement good practice guides for hot uniaxial and hot plane-strain compression testing [43, 51, 50, 46]:
  - Testing organisation information.
  - Test number and file number.
  - Material composition and batch number.
  - Details of testing procedure, including apparatus description and heating and cooling paths.
  - Data collection information.
  - Example calculations for processing of raw data.
  - Important results values, including maximum load, maximum flow stress, and final true strain.
  - Test validity measurements, for example the barrelling and ovality coefficients defined in equations (2.40) and (2.41).
  - Raw and processed curves for true stress, temperature, and strain-rate against true strain.
  - Raw and processed curves for force, temperature, and true strain against time.

Measurements from mechanical tests contain random error due to minor imperfections in calibration, geometry, and electronic noise [12]. Additionally, differences in composition, impurities, defects, and slight processing differences result in materials with the same technical designation exhibiting

different material behaviour and mechanical properties. These variations may be particularly significant if samples are taken from different batches, but it has also been shown to be significant for samples from the same batch [55].

To reduce uncertainty about the source of variation, the consistency of test results should be evaluated. For each testing condition, at least two specimens should be sampled from adjacent locations. If results differ by more than 5%, a third test should be performed [50, 43]. It may not be feasible to repeat tests for every combination of test conditions. In that case, a repeatability check can be performed by comparing results to those predicted by a constitutive model fitted to previous experiments from the same testing program.

An overview of mechanical testing techniques has been given in this section, and specific details were given for uniaxial tensile testing, plane-strain tension testing, uniaxial compression testing, and plane-strain compression testing. The importance of adhering to standardised protocols and best practices to ensure accurate and reliable test results was also highlighted. Some further mathematical details relating to mechanical testing are given in Appendix A.

In this chapter, the basics of modelling the behaviour metallic materials using plasticity equations, as well as details related to obtaining experimental data for the models, have been covered. The collated knowledge serves as a resource for the software design considerations presented in the next chapter.

3 Software Design

> Let things flow naturally forward in whatever way they like.
>
> *Lao Tzu*

SOFTWARE *design aspects of the project are discussed in this chapter. The chapter includes a breakdown of tasks for processing mechanical test datasets in Section 3.1, an outline of a strategy for handling such datasets in Section 3.2, and a list of software design requirements for processing a mechanical dataset according to the identified strategy in Section 3.3.*

3.1 Work Item Breakdown

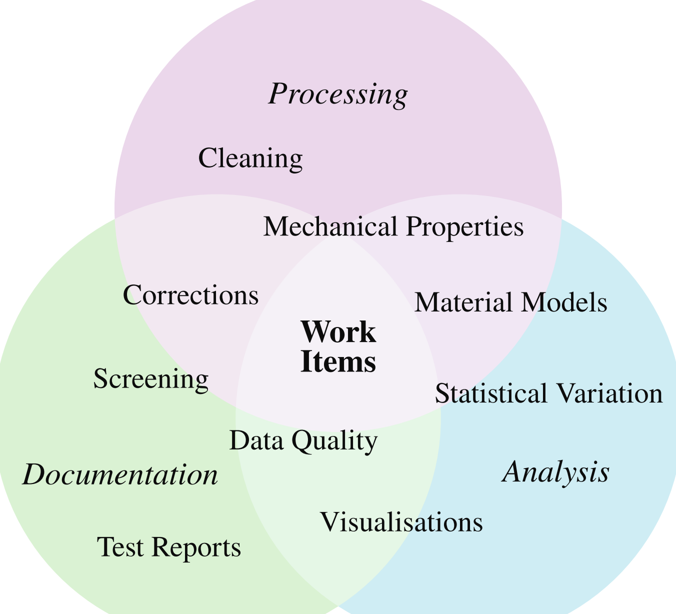


Figure 3.1: *Venn diagram illustrating the scope of work in processing mechanical test datasets, showing the overlap between the processing, documentation, and analysis categories.*

After reviewing the literature, a scope of work for processing mechanical test datasets could be

defined. This scope of work was categorized into three main areas: processing, documentation[1], and analysis. These categories are not mutually exclusive and contain shared topics, as shown in Figure 3.1.

Descriptions for these work items are given in Table 3.1. They represent steps necessary for valid and repeatable analyses of mechanical test data, specifically within the context of automated processing.

Table 3.1: *Descriptions of work items for processing a mechanical tests dataset.*

Work Item	Description
Cleaning	Ensuring the data adheres to a consistent format, and removing experimental artifacts and irrelevant data.
Corrections	Making appropriate adjustments to ensure the validity of the data, that is, ensuring that the data accurately reflects the behaviour of the material being tested.
Mechanical Properties	Identifying and quantifying key mechanical properties of the material; such as Young's modulus and yield strength.
Material Models	Fitting mathematical models that approximate the shape of the experimental measurements.
Statistical Variation	Analysing statistical variation in the dataset to determine the reliability and confidence in the determined mechanical properties, as well as the variability in material behaviour.
Visualizations	Creating visual representations of the data to aid in understanding and interpretation. This includes plots showing the stress-strain curves, as well as plots for showing the mechanical properties.
Data Quality	Reporting the degree to which the data is accurate, complete, consistent, and conforms to the relevant standards.
Test Reports	Generating reports that summarize the experimental setup, raw and processed test results, and key findings from the data analysis.
Screening	Semi-manually inspecting the raw and processed to identify any potential errors or anomalies.

Cleaning datasets is essential for automated processing. This involves ensuring that the data is stored in a consistent format, and eliminating undesired experimental artifacts. Removing experimental artifacts is necessary to isolate relevant data for analysis from irrelevant data. This prevents unnecessary memory usage and delays in automated processing.

Algorithms for automatically identifying mechanical properties, such as the offset yield strength, tensile strength, and fracture strength illustrated in Figure 2.3, are also more effective when applied to

[1] In this context, *documentation* refers to the collation of experimental information and data transformation history, and not to code documentation.

cleaned data. Other mechanical properties that might be determined by algorithmically identifying points of interest on the stress-strain curve include the Young's modulus and proportional limits. Identifying the proportional limits is a necessary step in applying foot correction, which entails shifting the data so that a line drawn through the elastic region of a stress-strain curve intersects the origin.

Corrections[1] are adjustments made to experimental data in order to ensure that it accurately reflects the behaviour of the material being tested. The purpose of these corrections is to ensure that the data is valid and measures what the experiment was intended to measure, taking into account any factors that might have affected the data. Examples of correction procedures other than foot correction are converting the engineering stress and strain to true stress and strain using, amongst others, equations (2.32) and (2.33), and correcting the data for thermal expansion using equation (2.46). Several other corrections procedures were also described in Section 2.2.

While automation can be used to facilitate the cleaning and correction of data, these steps still require a screening process to verify the results. For instance, fitting models can be done algorithmically by determining the material parameters that produce the minimum error. However, "visual inspection should still be performed to ensure that the resulting model has an appropriate general shape and inflection points" [56]. Screening is also a useful step for identifying individual failed experiments, and sanity checks should be employed during this step to ensure that results are physically realistic.

Screening can also be used to identify outliers. For this purpose, the statistical variation in results must be quantified. Statistical variation refers to the variation that occurs in any dataset due to random and uncontrollable factors. This variation can arise due to a variety of reasons such as measurement errors, environmental factors, or inherent variability in material composition. Understanding and quantifying this variation is important for accurate data analysis and interpretation, as well as for identifying potential outliers.

The screening process is crucial for maintaining the quality of data. Data quality refers to the degree to which data is accurate, reliable, complete and consistent. Accurate data is free from errors and reflects the true values of the measurements or observations. Reliable data is consistent and reproducible across different measurements or observers. Complete data contains all the information necessary to define the experiment. Consistent data is free from contradictions and adheres to a predefined set of rules or standards.

Ensuring data quality involves maintaining data documentation using test reports, using standardized data collection methods, and performing data validation and verification. Additionally, data cleaning, correction, and screening procedures help to maintain data quality.

A breakdown of the work items required for repeatable processing of a mechanical test dataset has

[1] Data correction is a somewhat misleading term as the underlying data is preserved and is then processed to create new data. This processing is usually performed so that the new data better represents the material properties.

been presented in this section. Material models and test reports items were covered in previous sections (Section 2.1.3 and Section 2.2.7). A detailed description of how these work items are applied during the processing of a mechanical test dataset is given in the next section.

3.2 Dataset Processing Strategy

Development of the software design was an iterative process that involved reviewing the literature and testing functionalities on the example datasets. This was done until the developed toolkit could be used to easily and efficiently repeat the analyses previously performed on the example datasets, as demonstrated in Chapter 5. By identifying common processing steps required for different datasets, a general strategy for handling mechanical test datasets could be developed.

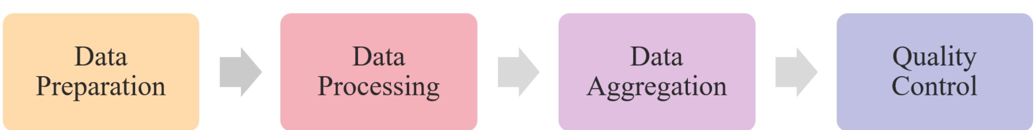


Figure 3.2: *Phases of the dataset processing strategy.*

The strategy consists of four phases displayed in Figure 3.2: data preparation, data processing, data aggregation, and quality control. These phases were further divided as follows:

- **Data Preparation:**

  1. *Gathering data and information.* Raw experimental measurements from the apparatus and information about the test-setup were retrieved. This was a different procedure for each dataset, and often required manual intervention, but the goal was always to collect the data and store it in a standardised format. The experimental measurements from each test were stored in a single data file. The metadata[1] for each set of tests was stored in a single spreadsheet. This file structure is illustrated in Figure 3.3.

  2. *Standardising file formatting.* The experimental measurements needed to be converted into a common format, which was chosen to be a CSV[2] file, in which each column was a vector of time-series measurements for the quantity described by the first row, or header row, see the data files on the left of Figure 3.3.

  3. *Applying a naming convention.* A naming convention was implemented for each dataset by allocating a unique string, or "test ID" to each test. The data files were named `test_ID.csv`. A column for test IDs was also added to the metadata

[1] Metadata is data about data. In this document, metadata refers to additional information about the experiment that produced the time-series measurements in the data files. Furthermore, mechanical properties and model parameters that are identified later are included in the metadata.

[2] A CSV (Comma Separated Values) file is a type of plain text file that stores tabular data.

spreadsheet. The row of metadata in the spreadsheet could be then easily matched to the relevant CSV file using the filename, as illustrated in Figure 3.3. Thus, the test ID in each row of the metadata spreadsheet could be used to identify the corresponding data file.

4. *Identifying groupings.* Unique categories of relevant information about the tests in each dataset were used to group the datasets. The number of items in each group could then be displayed using an experimental matrix, and the same groupings could be used to split the data into subplots to produce overview visualisations of the experimental data.

- **Data Processing:**

  1. *Trimming and cleaning.* Different methods, such as trimming using clustering algorithms[1]and trimming using sampling-rate changes, were applied to remove unnecessary leading and trailing measurements.
  2. *Applying corrections.* Correction procedures were applied to the cleaned data to improve its validity. This included corrections for friction and compliance-shift, as well as smoothing of oscillations from load-cell ringing[2].
  3. *Identifying points of interest.* Some values for mechanical properties could be identified from the data algorithmically. For those that could not be automatically determined, a simple GUI (Graphical User Interface) was developed and used to manually select points on the curve.

- **Data Aggregation:**

  1. *Making representative curves.* The previously identified groupings within the dataset were used here to combine measurements from groups of tests into mean curves with upper and lower bands. The bands were used to display statistics from the averaging, like standard deviations or maximum and minimum bounds. These representative curves could be used to display variation in the material behaviour, or to identify outliers.
  2. *Fitting models.* This involved curve-fitting[3], where time-series can be represented with optimised parameters for constitutive equations.

[1]The Gleeble 3800 testing apparatus, for example, produces displacement-force data with a cluster of points around the origin, which cannot be easily removed. When working with example datasets, it was found that machine learning clustering algorithms could be used to remove the unwanted data points effectively.

[2]See Section 2.2.6.

[3]See Section 2.1.4.

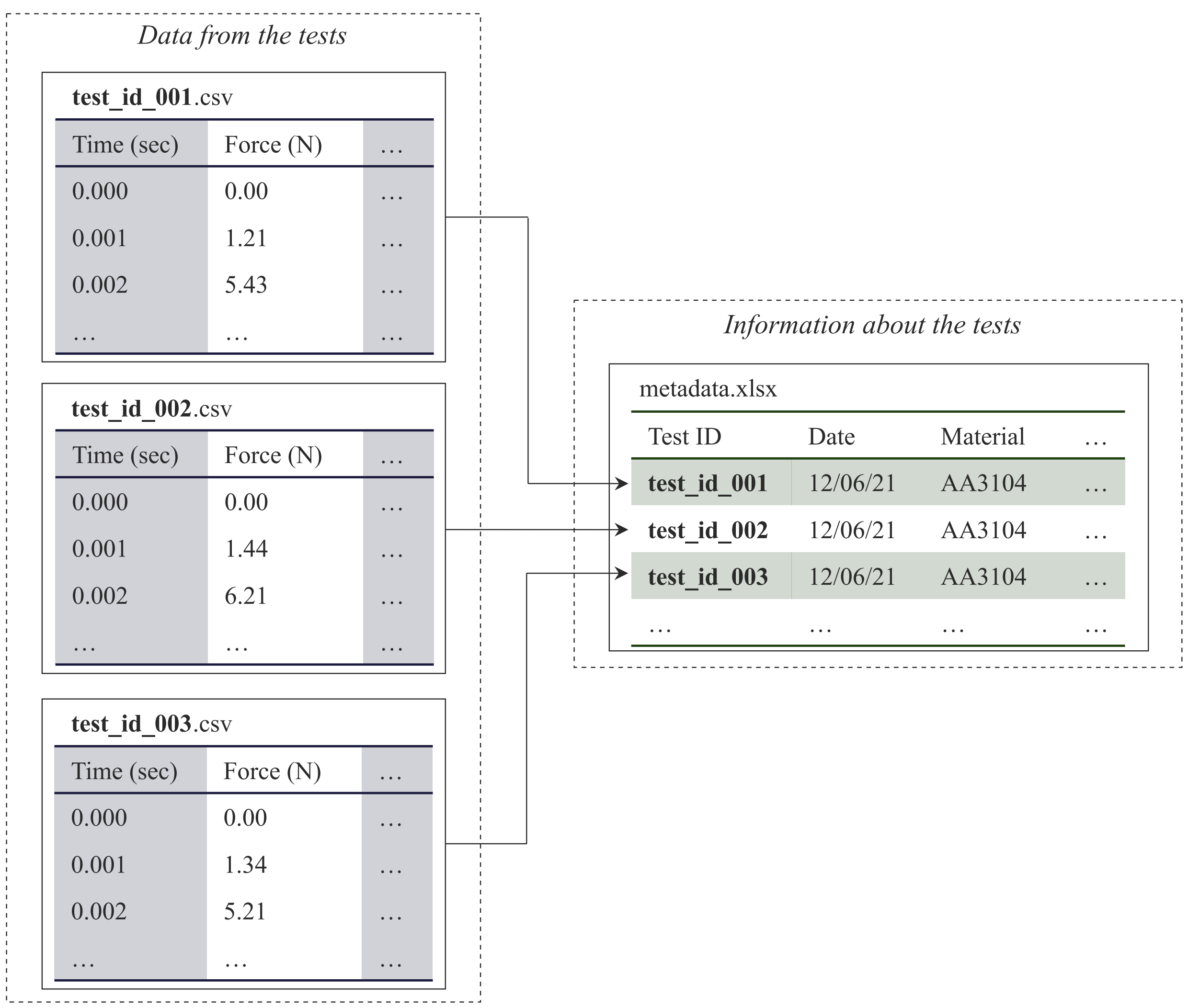


Figure 3.3: *Data storage file structure diagram. The data from each test is stored in a CSV file, left, and the information for each test is contained in a single row in the metadata spreadsheet, right. Each dataset consists of a set of data files, and a single metadata spreadsheet.*

Table 3.2: *Examples of metadata features for a processed mechanical test dataset.*

Metadata Category	Features
Test Information	Test ID, test date, experimentalist, apparatus, material, test type, specimen geometry, number of hits, total strain, nominal strain-rate, nominal temperature, sampling rate of measurement device, friction coefficient
Processing Variables	Trimming indices, proportional limits, elastic moduli, strain-shift values, yield stresses, proof stresses, 0.1 flow stresses
Model-Fitting Parameters	Parameter names, parameter units, fitting success/failure, parameter bounds, first guesses, optimised parameter values, fitting errors

- **Quality Control:**

  1. *Evaluating conformance to models.* Different groups of the datasets could be checked against the predictions from the fitted models. Using a conformance matrix, which showed the error for each subset, groupings with higher errors could be identified. This indicated that the groups of tests might need to be redone, or that the fitted models were not adequate to describe these groups.

  2. *Screening.* A semi-automated screening method was implemented, whereby a PDF with plots of data from individual tests on each page was generated. These pages included interactive fields for marking a test as accepted or rejected and for entering comments. By generating a screening PDF with this format for the entire dataset, users could quickly review the data plots for each test, and reject a test with a comment if necessary. The values entered in the fields could then be processed in bulk using programmed procedures based on the screening results.

  3. *Generating test reports.* Automated test reports were created to document the experimental data, processing history, and data quality for each test. This was done by creating a general LaTeX template file that included all relevant information and data visualizations on a single page for each test. These reports could be used as a reference for a specific test or as a more comprehensive screening step, without the interactive fields. The test reports were added to each dataset as they provided an overview of the data and detailed the processing steps, making it possible to trust the processed data and avoid repeating the processing.

  4. *Compiling dataset reports.* Jupyter Notebooks were used to create dynamic and easily checkable or adaptable dataset reports, which can be opened and run in a browser. These reports included tables of information and plots of data that showed the processing history. The lines of code used to perform the data processing procedures are included in these reports, as well as visualizations of the data after each processing step. This provides repeatability and traceability to the processed datasets.

Once data preparation, processing, aggregation, and quality control were completed, raw measurements and disorganised metadata had been transformed into a clean set of standardised data files and a table of metadata containing test information, mechanical properties, and fitted model parameters. The resulting feature-enriched table of metadata was ready for further statistical analysis. Specific examples of features contained in the final metadata table are outlined in Table 3.2.

The inputs and outputs for each phase of the strategy are summarised in Table 3.3. The phases are sequential and each phase is dependent on the completion of the previous phase. The inputs and outputs from each phase are also displayed in Figure 3.4, which provides a graphical overview of the processing strategy in the form of a flow-chart. The barrel-shaped icons on the left of the figure show the data and information being transferred between phases. The icons on the right of the figure show the outputs from each phase that can be used for analysis. Although the screening

Table 3.3: *Descriptions of inputs and outputs for each phase.*

Phase	Input	Output
Data Preparation	Output files from the apparatus containing raw measurements and additional recorded information about the test (metadata).	Set of standardised data files, with naming convention applied, and metadata spreadsheet.
Data Processing	Prepared test data files and metadata spreadsheet.	Data files with cleaned and corrected measurements, and metadata spreadsheet with added processing history and identified mechanical properties.
Data Aggregation	Files with cleaned and corrected test data and metadata spreadsheet with test and processing information.	Data files with representative curves, and metadata spreadsheet with added fitted parameters and errors from curve fitting.
Quality Control	Files with cleaned and processed test data and representative curves and metadata spreadsheet of test information, mechanical properties, and model parameters.	Files with processed and screened data, a metadata spreadsheet with added screening comments, test receipts, and a dataset report.

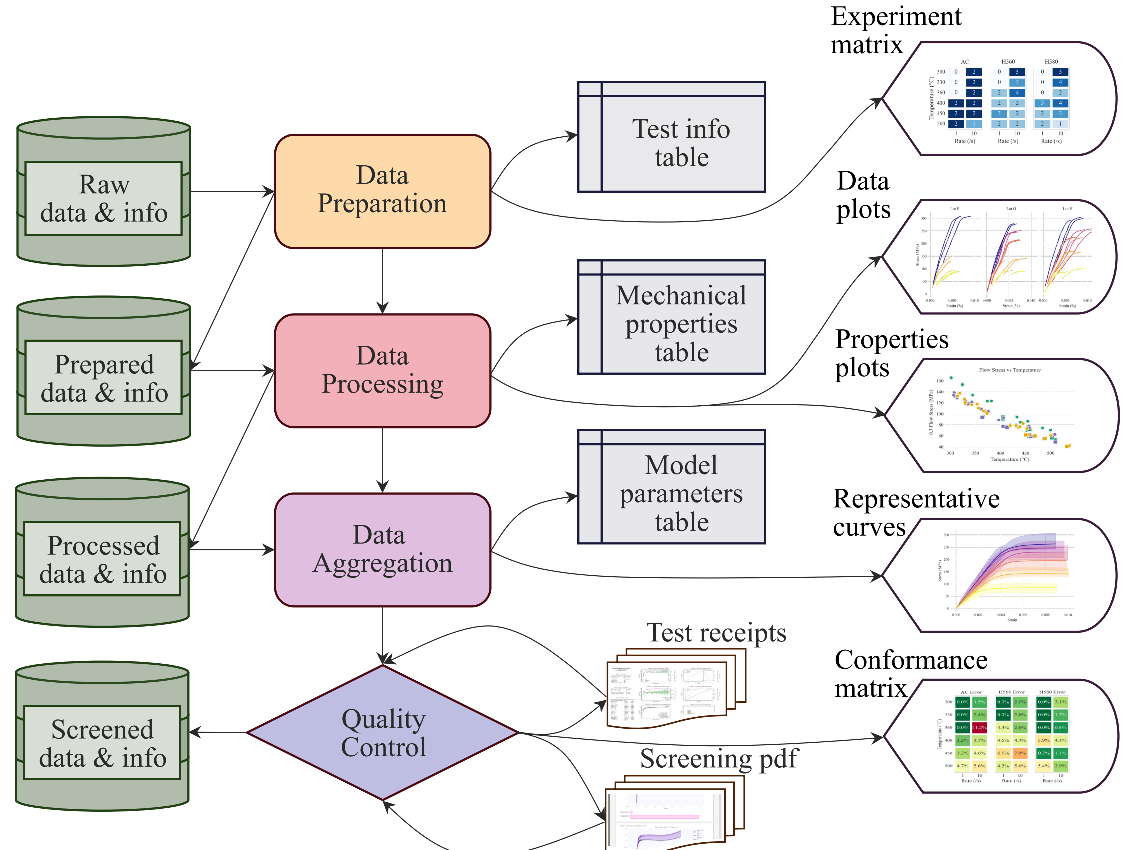


Figure 3.4: *Flow-chart showing the different phases of the use-case strategy.*

mechanisms (screening PDF and test reports) are linked only to the quality control phase in the flow-chart, they can be utilized in preceding phases. Furthermore, the quality control phase may identify issues that necessitate repeating tasks from earlier stages.

A strategy for processing mechanical test datasets has been presented in this section as a structured list of phases. The sequential nature of the strategy ensures that each phase is dependent on the completion of the previous one. Overall, the strategy offers a systematic approach to handling datasets.

3.3 List of Requirements

The strategy outlined in the previous section was used as a user-story[1] to develop the requirements presented in this section. These requirements are framework-independent and describe the steps necessary for processing a mechanical test dataset. They also serve as success criteria for the case studies, presented in Chapter 5, which should be examples of how to effectively process test datasets. The requirements address various aspects of dataset processing, such as data cleaning, feature extraction, and model selection:

- **(R1)** *Preparedness check.* The data files should be checked for uniqueness and to see if all data is formatted correctly. There should also be a check to confirm that the data files match the spreadsheet of metadata.
- **(R2)** *Batch processing.* The data should be processed in bulk using automation.
- **(R3)** *Grouping.* Groupings of similar and repeated tests in the dataset should be identified.
- **(R4)** *Overview visualisations.* The groupings should be used to produce overview visualisations of the data. These are useful for viewing processing results across the entire dataset.
- **(R5)** *Cleaning.* The data should be cleaned prior to further processing.
- **(R6)** *Corrections.* Correction procedures relevant to the test type and apparatus should be applied to the data.
- **(R7)** *Identifying mechanical properties.* Points of interest in the data that are used to calculate mechanical properties should be identified.
- **(R8)** *Fitting models.* Constitutive equations should be fit to the data to parameterise the curves.
- **(R9)** *Conformance evaluation.* The fitting error or conformance to fitted model of different tests and groups of tests should be evaluated.
- **(R10)** *Representative curves.* Representative curves should be generated for the identified groups of data. This includes determining statistics for the metadata features of the groups.

[1] A user-story is a description of features and functionalities from the perspective of an end-user.

(R11) *Screening.* Raw data and processed data should be manually screened to check that raw measurements and processing results appear valid.

(R12) *Test reports.* Test reports for each test should be generated. These should show the raw data, processing history, and any important analysis.

(R13) *Dataset report.* An overview of the prepared data, visualisations of the intermediate processed data and final processed data, identified mechanical properties, and fitted models for the entire dataset should be included in a combined dataset report. This should be done within a Jupyter Notebook, so that the lines of code used to process the data and produce visualisations are included, allowing for the entire procedure to be easily repeated or adapted.

Further requirements that are not related to the user-story, but are important for the development of the toolkit, include the following:

(R13) *Usability.* Tools for processing mechanical test datasets should be easy to use.

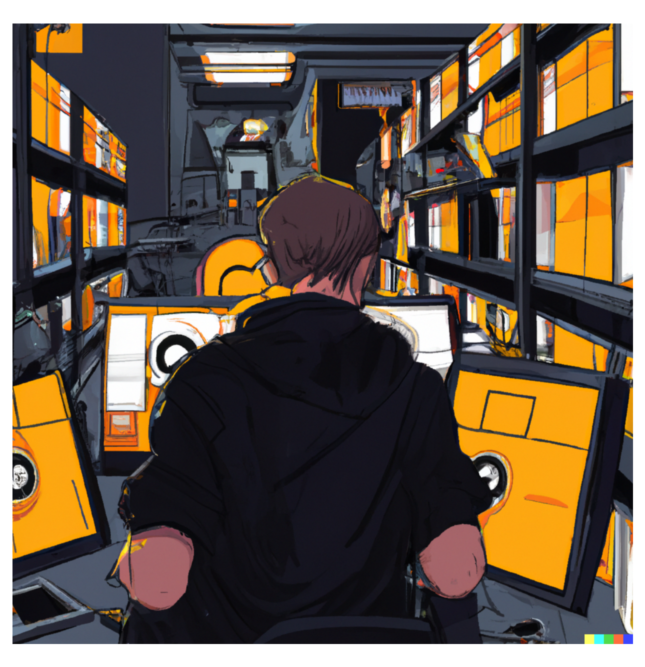
(R14) *Extensibility.* Tools for processing mechanical test datasets should be versatile and extensible.

The requirements highlight the necessity for adaptable software functionalities, making an open-source implementation preferable. Open-source software could be modified or extended to accommodate new functions tailored to the specific processing needs of various mechanical test types or material models. Users could actively contribute by adding features, identifying and fixing bugs, or suggesting enhancements. This collaborative approach enables the software to evolve in response to user needs.

Software design requirements for processing mechanical test datasets have been presented in this section. Various aspects were addressed, including data cleaning, feature extraction[1]*, model fitting, screening, and reporting. These requirements were informed by the work items and processing strategy presented in the previous sections of this chapter.*

The next chapter presents the implementation of a software package called Paramaterial. Paramaterial offers functionalities that fulfil the requirements outlined above. The open-source implementation of the package includes the necessary infrastructure for users to adapt and enhance it, ensuring that it will remain useful for addressing unforeseen future requirements.

[1]Features that can be extracted from mechanical test datasets include material-specific mechanical properties and constitutive model parameters. Values for these features can be found across a range of strains, temperatures and strain-rates, depending on the contents of the dataset.



4 Implementation and Usage

> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.
> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.
>
> *The Zen of Python*

THIS chapter contains an overview of the implementation and functionalities of Paramaterial, a Python package developed for the processing and analysis of mechanical test datasets. A summary of the package is given in Section 4.1 and a usage example is given in Section 4.2, before important algorithms are presented in Section 4.3.

4.1 Paramaterial

The layout of this section mimics that of a quick-start guide for a typical open-source Python package, but incorporates additional commentary to make it appropriate for this dissertation. The section is subdivided as follows: an overview of the package is provided in Section 4.1.1, the user-interface is discussed in Section 4.1.2, installation instructions and package dependencies are covered in Section 4.1.3, and instructions for contributing or logging issues are given in Section 4.1.4.

4.1.1 Overview of Paramaterial

Figure 4.1: *Paramaterial logo with badges showing current information about the package. This appears at the top of the README in the GitHub repo, and on the PyPI page.*

Part of developing an open-source software package is to make it attractive to potential users and contributors. Moreover, the project's purpose should be readily apparent at first glance. To aid with these goals, a logo was created that is intended to be visually appealling while hinting at the purpose of the software. The Paramaterial logo[1] is shown in Figure 4.1. This logo appears at the top of the project README on GitHub and PyPI.

Paramaterial is a powerful open-source software toolkit designed for parameterising materials test data. Paramaterial provides functionality for the repeatable processing of mechanical test results, such as stress-strain data from a tensile test. An example of various stages of data processing that might be performed using the toolkit is shown in Figure 4.2.

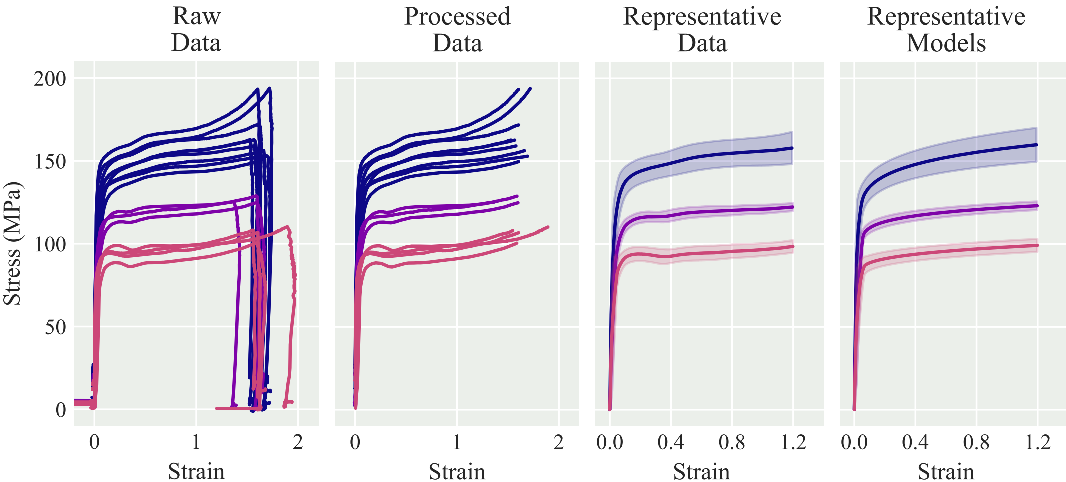
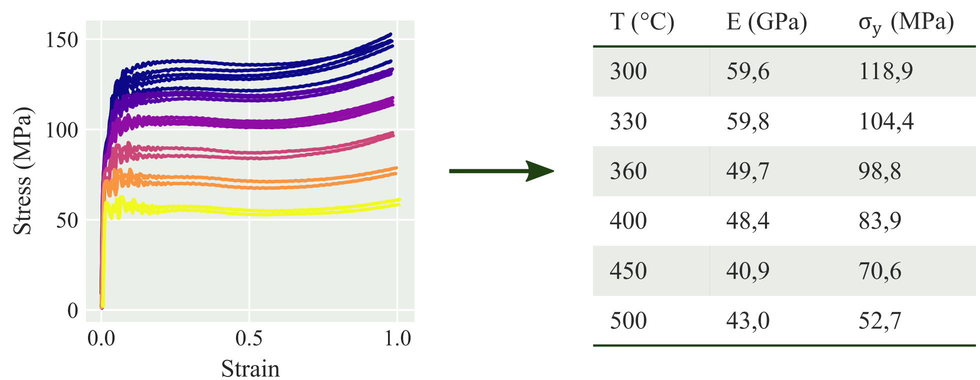


Figure 4.2: *Using Paramaterial to process large stress-strain datasets.*

Paramaterial is also useful for generating a table of parameters from raw data, as illustrated in Figure 4.3. Various data analysis techniques can then be applied to this table of parameters.



T (°C)	E (GPa)	σ_y (MPa)
300	59,6	118,9
330	59,8	104,4
360	49,7	98,8
400	48,4	83,9
450	40,9	70,6
500	43,0	52,7

Figure 4.3: *Using Paramaterial to make a table of parameters for analysis.*

[1] See Appendix B.6 for more details about the logo and badges.

Figures 4.2 and 4.3 provide a quick overview of what a user might do with Paramaterial. For more detailed examples of what can be achieved using the package, see the usage example in Section 4.2 and the case studies in Chapter 5.

Paramaterial is licensed under the MIT License, which allows for free and open use, modification, and distribution of the software for any purpose, including commercial use. Users should read the full text of the license before using or distributing the software.

Links to various online resources for Paramaterial are provided below:

Python Package: https://pypi.org/project/paramaterial/
Source Code: https://github.com/dan-slater/paramaterial
Code Documentation: https://dan-slater.github.io/paramaterial/

4.1.2 User Interface

The user-interface, that is, the Application Programming Interface (API), of Paramaterial was developed as a set of Python functions and classes that are intended to be used in a Jupyter Notebook environment. By providing an API, the functionality of the package is made available for integration into larger software systems.

An API (Application Programming Interface) is a set of protocols, routines, and tools for building software applications. It specifies how different software components should interact with each other, and provides a standardized way for external applications to access and use the functionality of the software. In the case of Paramaterial, the user interface API is a set of Python functions and classes that can be imported and used by other programs or scripts. This API provides a standardised way for users to interact with the package and perform the necessary data processing and analysis tasks.

The Jupyter Notebook environment allows for the integration of code, data, and visualizations in a single interactive document. This enables users to explore and analyse their mechanical test data in a reproducible and transparent way. Furthermore, the cross-platform compatibility of the Jupyter Notebook environment also allows for the Paramaterial API to be used on a variety of operating systems, including Windows, MacOS, and Linux.

4.1.3 Installation and Dependencies

The terminal can be used to install the latest version of Paramaterial with the command `pip install paramaterial`. Older versions can be installed by adding a version number to the command. For example, the version of the package used to perform the analyses presented in this document can be installed with `pip install paramaterial==0.1.0`. It is recommended that users check the list of requirements during installation.

Table 4.1: *Package descriptions for the Paramaterial dependencies.*

Package	Description
NumPy	NumPy is a library for numerical computing in Python. It provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays.
SciPy	SciPy is a library that extends NumPy by providing many additional functions for scientific computing. It includes modules for optimisation, integration, interpolation, Eigenvalue problems, signal processing, linear algebra, and more.
Pandas	Pandas is a library for manipulating and analysing data in Python. It provides a data frame object for storing and manipulating tables of data, and includes tools for reshaping, merging, and cleaning data, as well as for data visualization and statistical analysis.
Matplotlib	Matplotlib is a popular library for creating visualizations in Python. It provides a range of plotting tools, from simple line plots to 3D visualizations, and can be used to create a wide range of charts and graphs.
Seaborn	Seaborn is built on top of Matplotlib and provides a higher-level interface for creating statistical graphics in Python. It includes several built-in themes and colour palettes, and can be used to create complex visualizations with minimal code.
Jinja2	Jinja2 is a template engine for Python. It allows for a user to define templates for generating text output, which can include dynamic content such as variables and control structures.
ReportLab	ReportLab is a library for generating and manipulating PDF documents in Python. It provides tools for creating PDFs from scratch, as well as for modifying and combining existing PDFs. It includes support for a range of features, such as text, images, vector graphics, and interactive forms.
PyPDF2	PyPDF2 provides functions for reading, manipulating, and extracting information from PDF files, as well as performing operations such as merging, splitting, and encrypting PDF files.
OpenPyXL	OpenPyXL is designed to operate with Excel spreadsheets. It facilitates the tasks of reading, writing, and modifying Excel files.
Svglib	Svglib is a Python library for manipulating vector graphics files. It provides the ability to convert SVG files into other formats such as PDF, PNG or JPEG.
Tqdm	Tqdm is a Python package that displays a progress bar for an iterable object. It enables the user to track the progress of a loop or a task, providing the status of completion, estimated time remaining, and other helpful updates.

Paramaterial has several open-source Python libraries as dependencies. Descriptions for these dependencies are given in Table 4.1. The NumPy, SciPy, and Pandas packages make up the Python scientific and data science ecosystem. Matplotlib and Seaborn are used for generating visualisations. Jinja2, PyPDF2, and ReportLab are used for writing to and reading from PDFs. The OpenPyXL package is used for handling Excel files, and the Svglib package is used during generation of

plots and test reports. Finally, the Tqdm package is used to display a progress bar to indicate the remaining time for processing tasks.

4.1.4 Contributing or Logging Issues

Users can propose new features or changes by logging a pull request on the GitHub repository. The pull request should contain a detailed description of the proposed changes, along with any relevant code, tests, or documentation. The Paramaterial community welcomes collaboration with researchers who are interested in expanding the functionality of the software to support their work. By working together, users can collectively improve the toolkit, make it more versatile, and ensure that it meets the needs of a broader range of research projects.

Any bugs or problems can be reported[1] by logging an issue on the GitHub repository. The issue should provide a detailed description of the problem encountered, along with any relevant information, such as error messages or sample code.

Details for the Paramaterial package, similar to what would typically be found in the README of an open-source Python package, have been presented in this section. Some examples of the use-cases for Paramaterial were discussed, along with installation and contribution instructions. A more detailed usage example is given in the next section.

4.2 Usage Example

This is an example of how to use Paramaterial to process a dataset of uniaxial tensile test measurements. The example follows the strategy outlined in Section 3.2, consisting of four phases: data preparation, data processing, data aggregation, and quality control. These phases of the example are presented in Section 4.2.1, 4.2.2, 4.2.3, and 4.2.4, respectively.

4.2.1 Data Preparation Example

The example dataset of tensile test measurements was obtained from Mendeley Data and is available at DOI: 10.17632/rd6jm9tyb6.2. To begin, download the data and make a spreadsheet of metadata. The metadata spreadsheet can be stored as an XLSX or CSV file, but the data files must be stored as CSV files. Rename the data files according to the naming convention used in the `test id` column of the metadata spreadsheet. Alternatively, the example Notebook and files can be automatically downloaded and extracted using the code in Listing 4.1.

After downloading the data files, preparing the metadata spreadsheet, and applying a consistent naming convention, the data can be manipulated using the automated functionality provided by

[1] Before submitting a new issue, it is advisable to verify that the problem has not been previously reported or resolved by checking the repository's existing issues.

Listing 4.1: *Run this Python script to automatically download the example.*

```
import paramaterial as pam
pam.download_example('basic_usage')
```

Paramaterial. Listing 4.2 demonstrates how to import Paramaterial and print out the version number. It is good practice to display the version number at the top of the Jupyter Notebook, so that future researchers can use the same version of the package if they want to repeat the analysis presented in the Notebook.

Listing 4.2: *Import the Paramaterial package and print out the version number.*

```
# Import paramaterial as a module
import paramaterial as pam

# Check the version of paramaterial
print(pam.__version__)
```

Next, load the data files and metadata spreadsheet into a `DataSet` object. Listing 4.3 shows how to do this, as well as how to use a function provided by Paramaterial for checking the formatting of the data.

Listing 4.3: *Load the data and metadata into a DataSet object and check the formatting.*

```
# Load the metadata spreadsheet and data files into a DataSet object
prepared_ds = pam.DataSet('info/01 prepared info.xlsx', 'data/01 prepared data')

# Check the formatting of the loaded data and metadata
pam.check_formatting(ds=prepared_ds)
```

Once loaded, the dataset can be sorted using the `dataset.sort_by()` method. In Listing 4.4, the dataset is sorted by lot and temperature, before the `DataSet.info_table` attribute is used to get the current metadata[1]. The metadata is displayed in Table 4.2.

Listing 4.4: *Sorting the dataset and accessing the metadata.*

```
# Sort the dataitems in the dataset
prepared_ds = prepared_ds.sort_by(['temperature', 'lot'])

# Get the metadata as a table
prepared_ds.info_table
```

[1] The metadata spreadsheet that was loaded into the dataset is stored in the info_table attribute as a pandas.DataFrame object.

Table 4.2: *Metadata for prepared dataset.*

test_id	test_type	temperature	lot	number	rate
T_020_A_1	T	20	A	1	0.000866
T_020_A_2	T	20	A	2	0.000866
T_020_A_3	T	20	A	3	0.000866
T_020_B_1	T	20	B	1	0.000866
...	...	...	...	...	...
P_300_I_2	P	300	I	2	0.000866
P_300_I_3	P	300	I	3	0.000866
T_300_I_1	T	300	I	1	0.000866

The columns in Table 4.2 describe metadata categories for the prepared example dataset. The two test types, uniaxial tension and plain-strain tension, are respectively denoted by `T` and `P` in the `test_type` column. However, only the uniaxial tension tests are of interest for this example. The dataset can be filtered to remove the plain-strain tension tests using the `DataSet.subset()` method, as shown in Listing 4.5.

Listing 4.5: *Filtering the dataset to extract a subset.*

```
1 # Get a subset of only the tensile tests
2 prepared_ds = prepared_ds.subset({'test_type': ['T']})
```

Additionally, there are differing entries in the `temperature` and `lot` categories of the metadata. The `experimental_matrix` function can be used to view the distribution of tests in the dataset across these categories as a heatmap. The `experimental_matrix` function is demonstrated in Listing 4.6 and the resulting heatmap is displayed in Figure 4.4.

Listing 4.6: *Making an experimental matrix heatmap plot using Paramaterial.*

```
1 # make a heatmap showing the distribution across lot and temperature
2 pam.experimental_matrix(info_table=prepared_ds.info_table, index='temperature',
3                         columns='lot', as_heatmap=True)
```

The experimental matrix is a useful tool for identifying groupings in the dataset. These groupings can then be used to define a plotting function which produces a set of subplots, with one subplot per grouping.

Prior to plotting the figures, it is useful to instantiate a `Styler` object, which stores plotting formatters that later get passed into the Paramaterial plotting functions. The `Styler.style_to()` method sets up the `Styler` to match the metadata in the inputted dataset. This is demonstrated

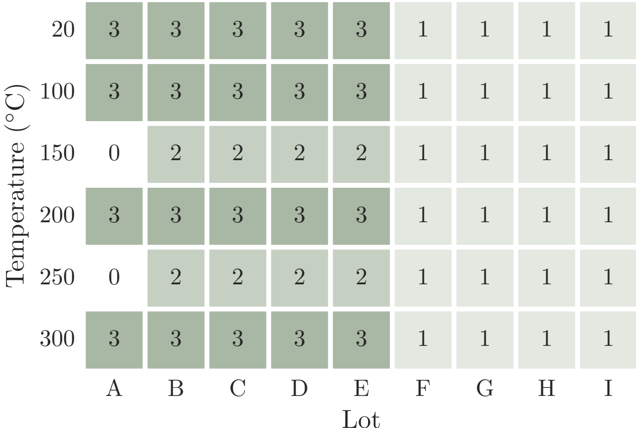


Figure 4.4: *Experimental matrix showing distribution of tests across temperature and lot.*

in Listing 4.7.

Listing 4.7: *Setting up a Styler object and labels dictionary to be used when defining plotting functions.*

```
1  # Instantiate a Styler object and format to match the prepared data
2  styler = pam.Styler(color_by='temperature', color_by_label='(°C)', cmap='plasma')
3  styler.style_to(ds=prepared_ds)
4
5  # Keys labels for stress-strain data plotting
6  labels_dict = dict(x='Strain', y='Stress_MPa', ylabel='Stress (MPa)')
```

The `styler` and `labels_dict` objects created in Listing 4.7 are subsequently used to set up multiple plotting functions. This approach reduces code repetition and helps to ensure uniform formatting across all the plots generated during the data processing example.

In Listing 4.8, the `styler` and `labels_dict` objects are used with the `dataset_plot()` function (from the Paramaterial plotting library) to define a new function, `ds_plot()`, for plotting the entire dataset on a single plot. The newly defined `ds_plot()` function is then also called in Listing 4.8, which produces the plot displayed in Figure 4.5.

Listing 4.8: *Defining a function that plots the stress-strain curves from the entire dataset on a single plot.*

```
1  # Define a function to plot the dataset
2  def ds_plot(ds, **kwargs):
3      """Returns a matplotlib Axes object."""
4      return pam.dataset_plot(ds=ds, styler=styler, **labels_dict, **kwargs)
5
6
7  # Plot the dataset
8  ds_plot(prepared_ds)
```

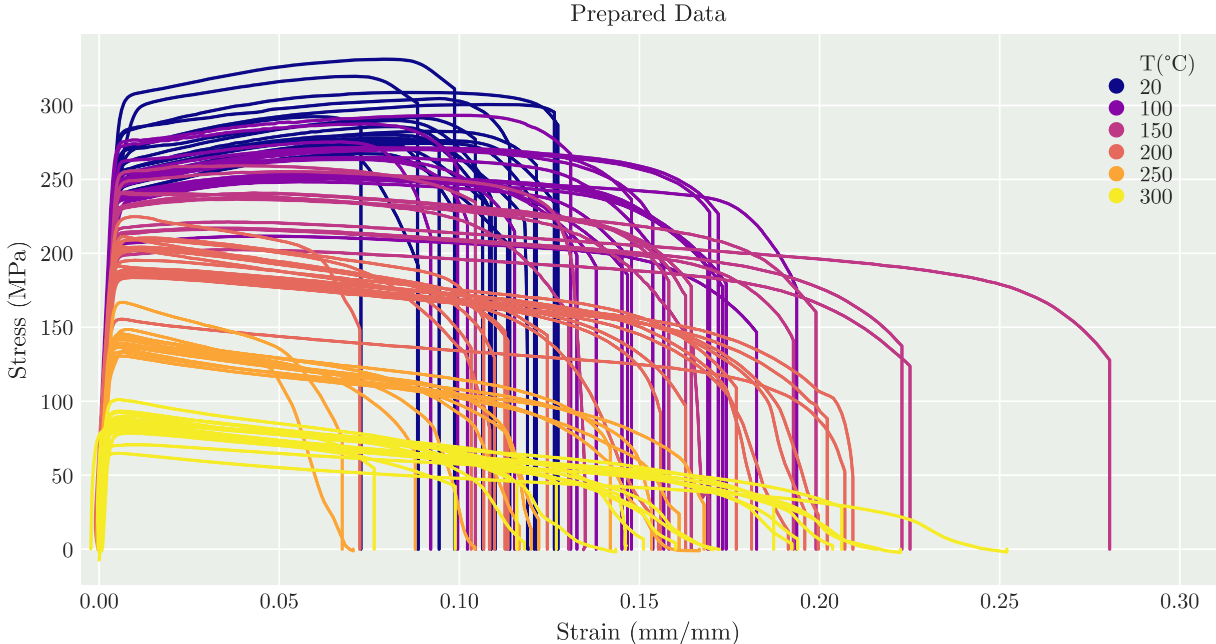


Figure 4.5: *Stress-strain curves from the prepared uniaxial tension test data.*

The `styler` and `labels_dict` objects are then used in Listing 4.9 to produce a further plotting function. Here, they are used with another of Paramaterial's in-built functions, the `dataset_subplots()` function, to define a new function, `ds_subplots()`. The `ds_subplots()` function produces the grid of subplots displayed in Figure 4.6. Note that the subplots function only displays data from lots A-E, and the tests on lots F-I are omitted.

The user-defined `ds_plot` and `ds_subplots` can then be used interchangeably throughout the ensuing analysis to visualise the data after various processing steps. Defining the plotting functions in this way allows for the user to maintain consistent formatting throughout the Jupyter Notebook.

Listing 4.9: *Defining a function that plots the stress-strain curves on a grid of subplots.*

```
1  # Define a function to plot the dataset as a grid of plots
2  def ds_subplots(ds, **kwargs):
3      """Returns an array of matplotlib Axes objects."""
4      return pam.dataset_subplots(
5          ds=ds, shape=(6, 5), styler=styler,
6          rows_by='temperature', row_vals=[[20], [100], [150], [200], [250], [300]],
7          cols_by='lot', col_vals=[['A'], ['B'], ['C'], ['D'], ['E']],
8          col_titles=[f'Lot {lot}' for lot in 'ABCDE'], **labels_dict, **kwargs)
9
10
11 # Plot the dataset as a grid of plots
12 ds_subplots(prepared_ds)
```

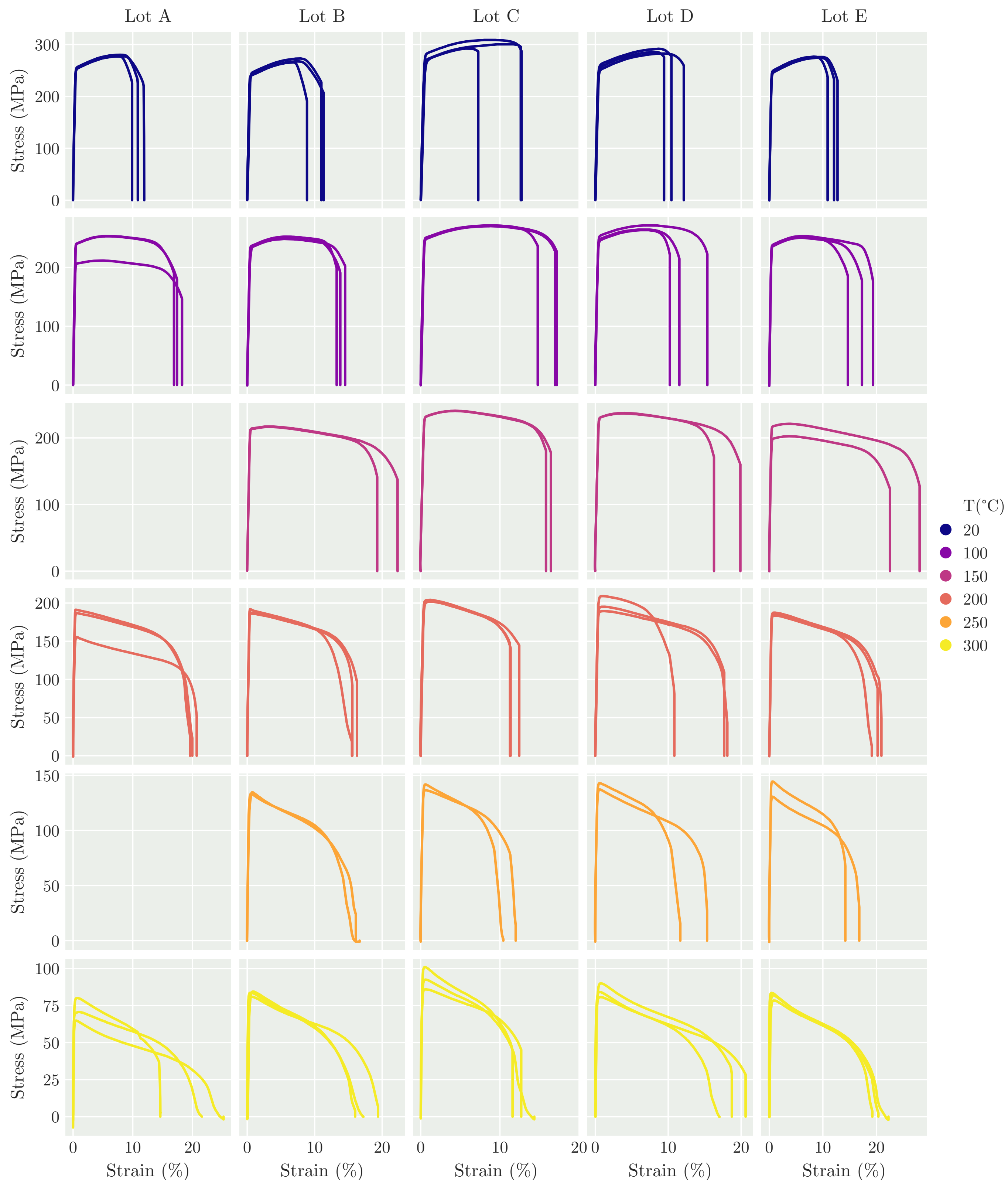


Figure 4.6: *Stress-strain curves from the prepared uniaxial tension test data for lots A to E and temperatures 20 °C to 300 °C, with a column of subplots for each lot and a row subplots for each temperature.*

4.2.2 Data Processing

The data processing phase is initiated after data processing. In this example, data processing entails finding mechanical properties, trimming, applying a correction, and a screening step for the correction.

Some of the mechanical properties are more easily obtained from the full curves, while for others it is more convenient to use the trimmed curves. Hence, before trimming to small strain, the ultimate tensile strength and fracture point of the stress-strain curves should be determined, as demonstrated in Listing 4.10.

Listing 4.10: *Automatically finding the ultimate tensile strength and fracture point.*

```
# Determine the ultimate tensile strength for each test
prepared_ds = pam.find_UTS(ds=prepared_ds, strain_key='Strain',
                           stress_key='Stress_MPa')

# Determine the fracture point for each test
prepared_ds = pam.find_fracture_point(ds=prepared_ds, strain_key='Strain',
                                      stress_key='Stress_MPa')
```

The `DataSet.apply()` method is used along with a user-defined trimming function in Listing 4.11 to trim all of the stress-strain curves in the `DataSet`. The user-defined function in Listing 4.11 only needs to be set up to operate on a single `DataItem`, and this operation is then applied to the entire `DataSet`. The trimmed curves are shown in Figure 4.7.

Listing 4.11: *Defining a trimming function and applying it to a dataset.*

```
# Define a function to trim the data to a small strain range
def trim_to_small_strain(di):
    """Takes in a DataItem and returns a DataItem."""
    di.data = di.data[di.data['Strain'] < 0.01]
    return di


# Apply the function to the dataset
trimmed_ds = prepared_ds.apply(trim_to_small_strain)

# Plot the trimmed dataset
ds_plot(trimmed_ds)
```

The trimmed data evidently requires foot correction. Paramaterial provides a function for identifying the limits of the elastic region, `find_upl_and_lpl()`, and a separate function for applying the foot correction, `correct_foot()`. The usage of these functions is demonstrated in Listing 4.12, and the resulting corrected data is displayed in Figure 4.8.

Listing 4.12: *Identifying the proportional limits and applying foot correction.*

```
1  # Determine the upper and lower proportional limits for each test
2  trimmed_ds = pam.find_upl_and_lpl(ds=trimmed_ds, preload=36,
3                                    preload_key='Stress_MPa')
4
5  # Apply the foot correction to the dataset
6  corrected_ds = pam.correct_foot(ds=trimmed_ds, LPL_key='LPL', UPL_key='UPL')
7
8  # Plot the corrected dataset
9  ds_plot(corrected_ds)
```

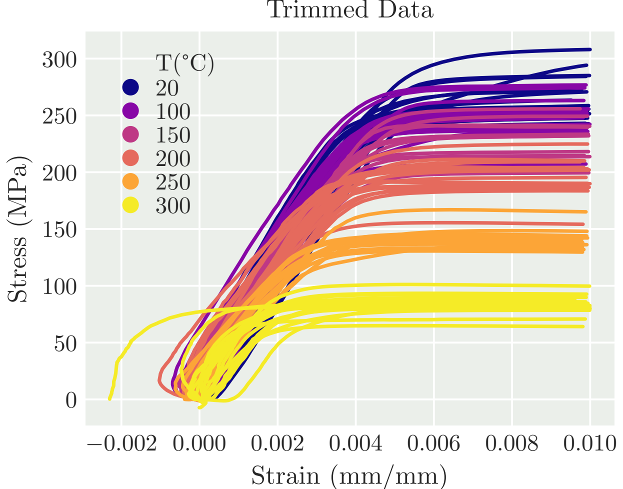


Figure 4.7: *Trimmed example data.*

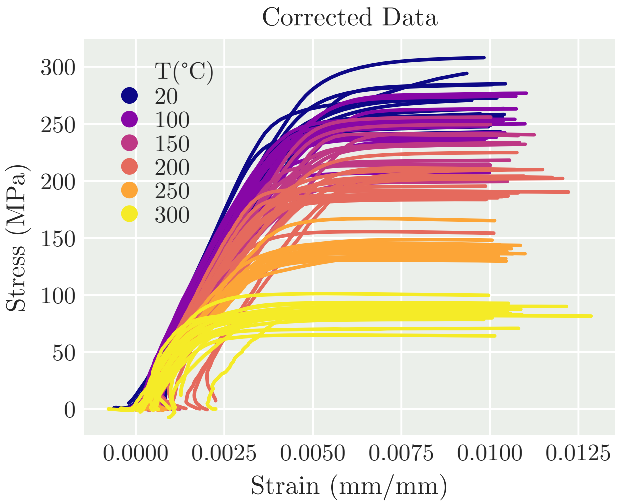


Figure 4.8: *Example data after foot correction.*

While the ends of the trimmed curves in Figure 4.7 are aligned, the ends of the corrected curves in Figure 4.8 are misaligned. This misalignment is evidence of the shifting that occurred during foot correction. The foot correction procedure thus appears to have worked for most of the example data.

However, several of the 200 °C and 300 °C curves appear to have been shifted too far to the right of the origin.

The Paramaterial library includes functionality for creating a screening PDF for manual intervention that can be used here to identify which of the curves where shifted too far to the right. To utilize the screening PDF, a user must define a function that produces a plot for a single test. The user-defined plotting function is then passed to the `make_screening_pdf()` function provided by Paramaterial, along with the path where the screening PDF should be saved and the dataset object containing the test data for the plots.

The resulting screening PDF contains a single page for each test, which presents the user-specified plot and test ID for that test, as well as a checkbox for indicating rejection and a textbox where users can enter their comments. An example of how to define a screening plot and generate a screening PDF is given in Listing 4.13, and a screenshot from the resulting screening PDF is displayed in Figure 4.9.

Listing 4.13: *Generating a screening PDF.*

```python
# Define a function to plot a single test for foot correction screening
def foot_correction_screening_plot(di):
    """Takes in a DataItem and returns a matplotlib Axes object."""
    color = styler.color_dict[di.info['temperature']]
    LPL = (di.info['UPL_0'], di.info['UPL_1'])
    UPL = (di.info['LPL_0'], di.info['LPL_1'])
    ax = ds_plot(corrected_ds.subset({'test_id': di.test_id}))
    ax = ds_plot(trimmed_ds.subset({'test_id': di.test_id}), alpha=0.5, ax=ax)
    ax.axline(UPL, slope=di.info['E'], c=color, ls='--', alpha=0.5)
    ax.plot(*UPL, c=color, marker=4)
    ax.plot(*LPL, c=color, marker=5)
    return ax


# Make the screening pdf
pam.make_screening_pdf(corrected_ds, plot_func=foot_correction_screening_plot,
                       pdf_path='info/foot-correction screening.pdf')
```

The screening PDF offers a user-friendly environment for swiftly perusing the outcome of an automated procedure on each test. By selecting the check-box, a specific test can be marked as invalid, and the comment box allows for the addition of an explanation for the rejection.

For instance, in Figure 4.9, the upper and lower proportional limits are in close proximity, which indicates that the elastic region was not properly identified. Therefore, this test was rejected.

Although a graphical user interface (GUI) has been incorporated in Paramaterial to manually

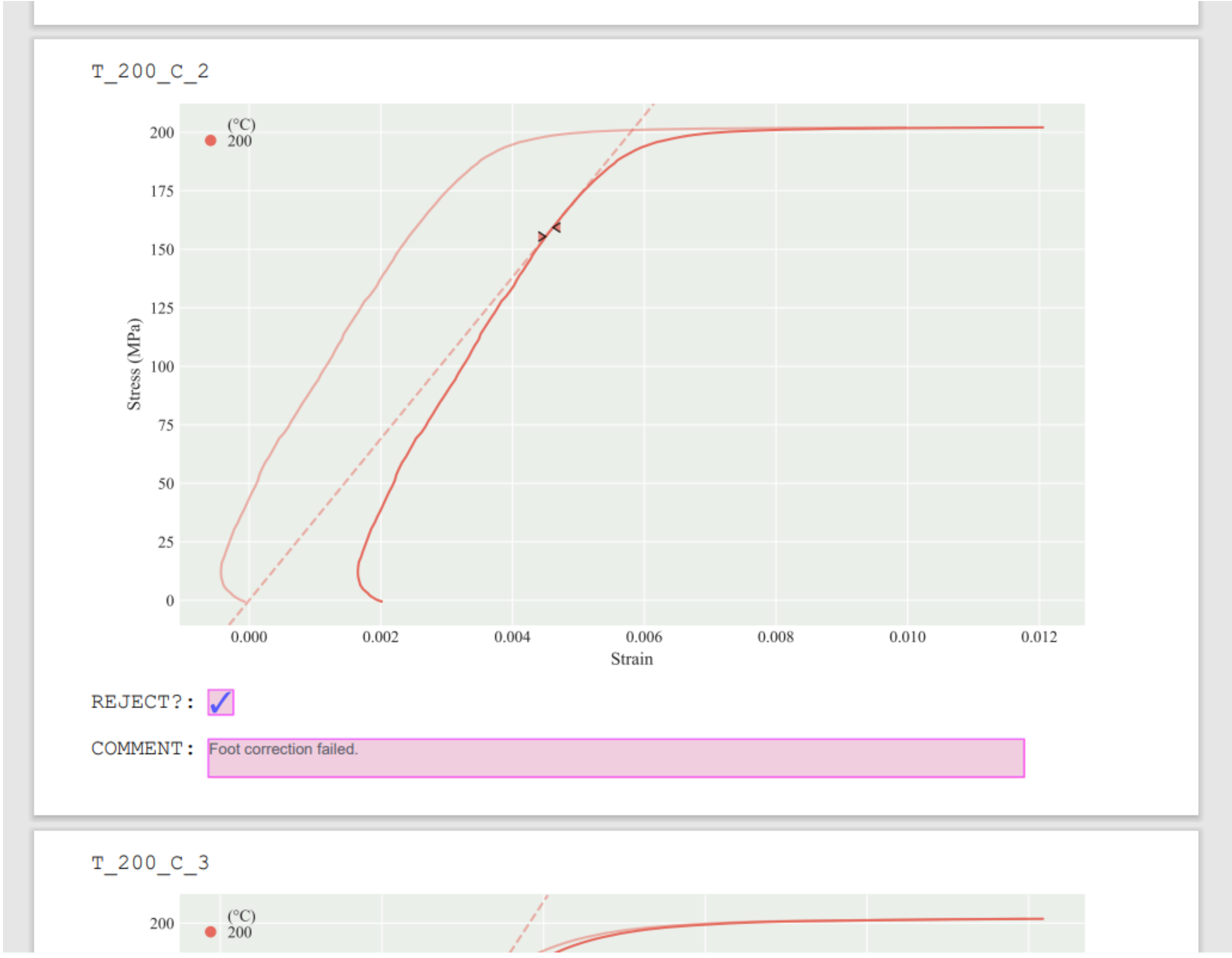


Figure 4.9: *Screenshot from the screening PDF used in the example.*

identify points of interest[1] such as proportional limits, such a method diminishes the repeatability of the analysis. Consequently, in this example, tests where the foot correction procedure failed are simply rejected, instead of using an alternative method to identify the elastic regions.

After the screening PDF has been annotated by selecting the check-boxes and by entering comments into the text-boxes, the entries in these fields can be imported into the `info_table` of a dataset using the code in Listing 4.14. A table of the rejected tests can then be obtained, which is displayed in Table 4.3.

Listing 4.14: *Reading the entries from a screening PDF into a DataSet object.*

```
1  # Read the annotated screening pdf fields to the dataset
2  corrected_ds = pam.read_screening_pdf(ds=corrected_ds,
3                                        pdf_path='info/screening-marked.pdf')
4
5  # Get the metadata table of the rejected items
6  corrected_ds.info_table[corrected_ds.info_table['reject'] == 'True']
```

[1] See Appendix B.5.

Table 4.3: *Tests that were rejected during screening.*

test_id	temperature	lot	number	reject	comment
test_ID_107	200	A	3	True	Foot correction failed
test_ID_112	200	C	2	True	Foot correction failed
test_ID_115	200	D	2	True	Foot correction failed
test_ID_121	200	G	1	True	Foot correction failed
test_ID_137	300	A	2	True	Foot correction failed
test_ID_146	300	D	2	True	Foot correction failed
test_ID_152	300	G	1	True	Foot correction failed
test_ID_153	300	H	1	True	Foot correction failed

The tests that have been flagged for rejection can then be removed from the dataset using the code in Listing 4.15. The resulting `DataSet` contains trimmed, corrected and screened curves.

Listing 4.15: *Rejecting tests that were flagged during screening.*

```
# Remove the rejected tests from the dataset
screened_ds = pam.remove_rejected_items(ds=corrected_ds, reject_key='reject')

# Plot the screened dataset
ds_plot(screened_ds)
```

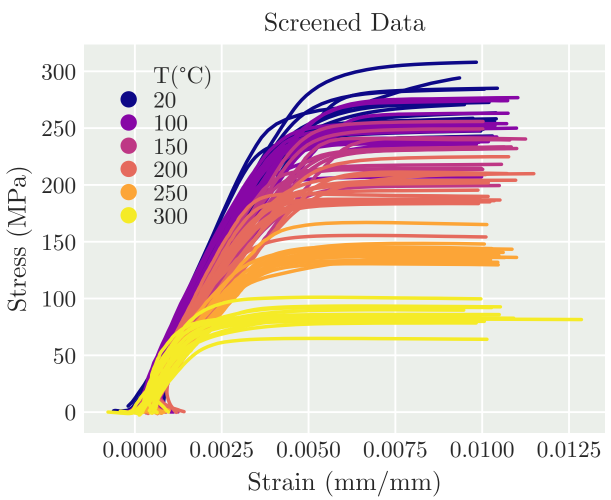


Figure 4.10: *Screened and foot-corrected example data.*

The screening results can be seen by comparing the corrected data in Figure 4.8 with the screened data in Figure 4.10; curves that were shifted too far to the right have been removed.

The proof stresses for each of the curves can be then be determined using the code in Listing 4.16. This is the last processing step for the example.

Listing 4.16: *Determining the proof stresses for a dataset of stress-strain curves.*

```
1  # Find the 0.2% proof stress for each test
2  screened_ds = pam.find_proof_stress(ds=screened_ds, proof_strain=0.2, E_key='E',
3                                      strain_key='Strain', stress_key='Stress_MPa')
```

It is advisable to save the processed data, as demonstrated in Listing 4.17. Doing so creates a checkpoint outside of the Notebook that captures the analysis progress. This approach also provides a clear distinction between the original raw data and the processed data files.

The `ds_subplots()` function, defined earlier in Listing 4.9, is then used again in Listing 4.17 to visualise the processed data. This produces the grid of subplots displayed in Figure 4.11, and allows the user to view and compare individual processed curves, before moving on to the data aggregation phase.

Listing 4.17: *Saving the processed data and visualising it on a grid of subplots.*

```
1  # Write the metadata table data files to the specified paths
2  screened_ds.write_output('info/02 processed info.xlsx',
3                           'data/02 processed data')
4
5  # Load the processed dataset
6  processed_ds = pam.DataSet('info/02 processed info.xlsx',
7                             'data/02 processed data')
8
9  # Plot the processed dataset as a grid of plots
10 ds_subplots(processed_ds)
```

4.2.3 Data Aggregation

Data aggregation is the process of summarizing a vast dataset into a more concise and comprehensive format. In mechanical test data processing, data aggregation often entails determining representative stress-strain curves, as well finding optimal values for material models.

Representative curves, which illustrate the average behaviour of the group of curves they represent, can be generated from groupings of the processed data. The `make_representative_data()` function provided by Paramaterial accepts an existing `DataSet` and produces a new one containing the desired representative curves. Additionally, this function consolidates the associated metadata from individual tests, such as yield strengths and proportional limits, allowing for the computation of statistical parameters, such as means and deviations. Usage of this function is demonstrated in Listing 4.18.

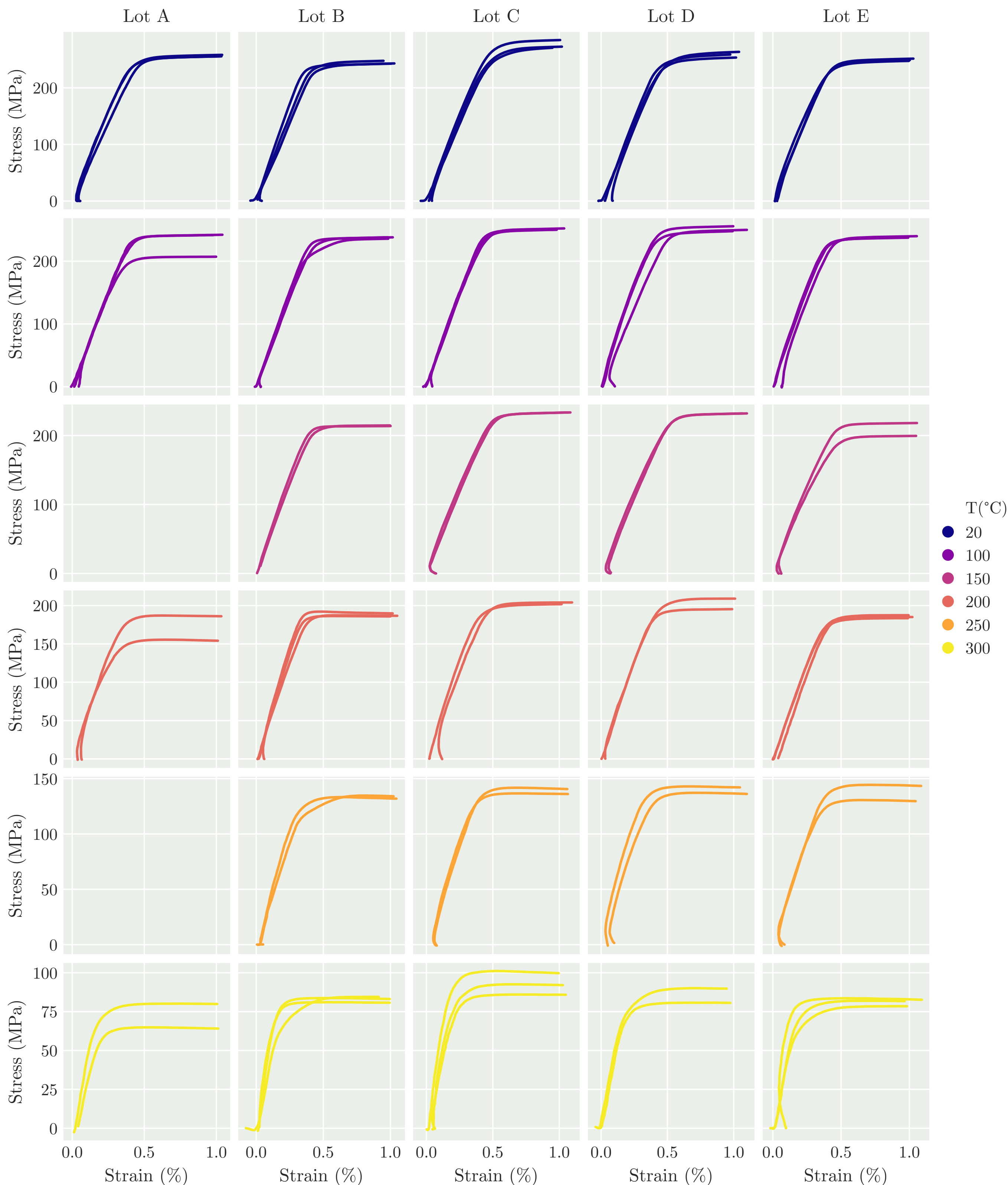


Figure 4.11: *Processed stress-strain curves for lots A to E and temperatures* 20 °C *to* 300 °C*, with a column for each lot and a row for each temperature.*

The `ds_plot()` function is also used in Listing 4.18 to plot the representative curves, resulting in the plot displayed in Figure 4.12. The representative curves show the means of the processed data temperature groups, with bands for the standard deviations. The associated representative metadata is given Table 4.4.

Listing 4.18: *Making representative data according to temperature groupings.*

```
# Make representative curves for each temperature and test type
pam.make_representative_data(
    ds=processed_ds, info_path='info/03 repres info.xlsx',
    data_dir='data/03 repres data', repres_col='Stress_MPa',
    interp_by='Strain', interp_range='inner',
    group_by_keys=['temperature', 'test_type'],
    group_info_cols=['UTS_0', 'UTS_1', 'FP_0', 'E',
                     'PS_0.002_0', 'PS_0.002_1', 'UPL_1'])

# Load the representative dataset
repres_ds = pam.DataSet('info/03 repres info.xlsx',
                        'data/03 repres data',
                        test_id_key='repres_id')

# Plot representative curves with standard deviation bands
ds_plot(repres_ds,
        fill_between=('down_std_Stress_MPa', 'up_std_Stress_MPa'))

# Get the metadata table of the representative dataset
repres_ds.info_table
```

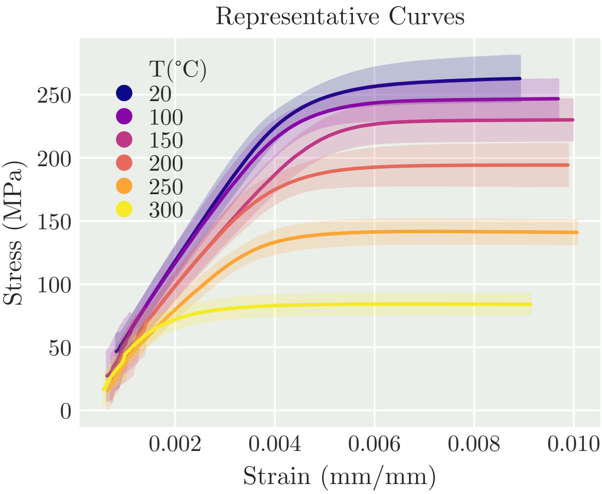


Figure 4.12: *Representative curves for the processed data grouped by temperature. Mean curves are shown with a band for standard deviation.*

The representative curves show the average material behaviour for each temperature. To capture this averaged material behaviour in a model that can be used to predict the material behaviour under different loading conditions, a constitutive equation curve must be fit to the data.

Table 4.4: *Representative info table.*

repres_id	temperature	E	std_E	...	PS_0.002_1	std_PS_0.002_1
repres_id_01	20	59652.66	6320.84	...	258.87	17.01
repres_id_02	100	59855.78	9214.04	...	244.37	15.45
repres_id_03	150	49741.58	5106.77	...	228.80	16.36
repres_id_04	200	49769.77	5853.61	...	193.26	15.87
repres_id_05	250	40860.57	5488.52	...	140.64	9.84
repres_id_06	300	47232.56	10162.62	...	82.88	8.22

Paramaterial offers the `ModelSet` class for fitting material models to data. The `ModelSet` class can be employed to fit a model to a `DataSet` class. However, before fitting, a model function that generates stress-data from strain-data must be defined. Paramaterial provides several built-in models, which can be imported from the `Paramaterial.models` module. For this example, the built-in Ramberg-Osgood model was fitted to the data, as demonstrated in Listing 4.19.

Listing 4.19: *Fitting a material model to a dataset using Paramaterial.*

```
# Get the Ramberg-Osgood model function
ramberg_func = pam.models.ramberg

# Set up the modelset, variables are known, params are unknown
ramberg_ms = pam.ModelSet(
    model_func=pam.models.ramberg,
    var_names=['E', 'UPL_1'],
    param_names=['K', 'n'],
    bounds=[(0., 1000.), (0.01, 0.8)]
)

# Fit the modelset to the representative dataset
ramberg_ms.fit_to(repres_ds, x_key='Strain', y_key='Stress_MPa')

# Get a table of the fitted parameters
ramberg_ms.fitting_results
```

In Listing 4.19, the `var_keys` argument specifies which model input arguments should be treated as variables. Values for these variables will be extracted from the existing metadata for each test during fitting. In this example, the `E` and `UPL_1` represent the Young's modulus and upper proportional limit, respectively. The model parameters, `K` and `n`, represent the hardening modulus and and strain-hardening exponent from equation (2.15). Once the model has been fitted to the dataset, the fitted parameters can be accessed using the `ModelSet.fitting_results` property. The resulting fitted parameters and are given in Table 4.5.

Once the optimal set of model parameters has been determined for each test in the provided `DataSet`, a `ModelSet` object can be utilized to create a new `DataSet` object containing synthetic data. This

Table 4.5: *Fitted parameters table.*

model_id	temperature	E	UPL_1	K	n	error
model_id_01	20	59652.66	163.25	216.58	0.14	5.99
model_id_02	100	59855.78	145.23	204.98	0.13	23.41
model_id_03	150	49741.58	130.89	177.94	0.10	15.66
model_id_04	200	49769.77	127.85	138.85	0.13	11.02
model_id_05	250	40860.57	86.39	97.18	0.10	9.69
model_id_06	300	47232.56	53.17	67.72	0.15	4.39

synthetic data, that is, stress-strain data obtained not from a physical test but rather from sampling a calibrated constitutive model, comprises a set of predicted curves that correspond to the respective measured curves from the input data. The `ModelSet.predict()` function generates the artificial dataset, as illustrated in Listing 4.20. The fitted curves are subsequently plotted on the same axis as the measured curves, resulting in Figure 4.13.

Listing 4.20: *Generating a dataset of predicted data using a fitted modelset.*

```
# Predict a dataset from the modelset
ramberg_ds = ramberg_ms.predict()

# Plot the fitted curves above the representative curves
ax = ds_plot(repres_ds, alpha=0.5)
ds_plot(ramberg_ds, ls='--', ax=ax)
```

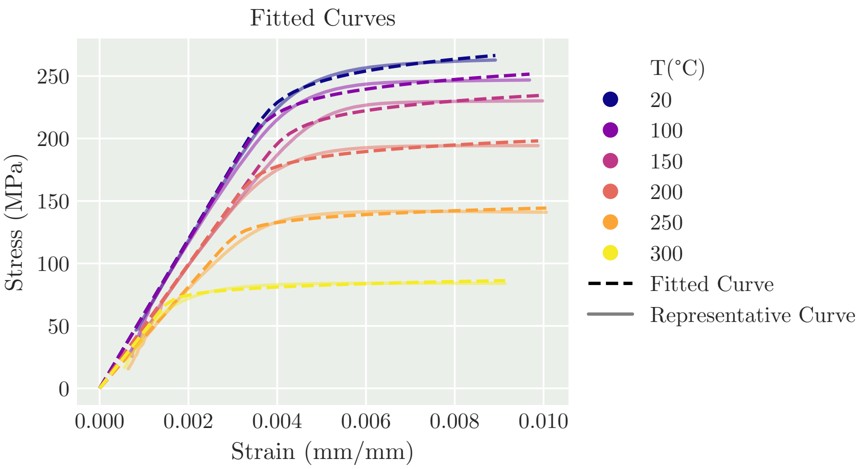


Figure 4.13: *Ramberg-Osgood model curves, dashed, plotted above the associated representative curves.*

After implementing the aforementioned processing steps, the dataset has been compiled into tables containing mechanical properties, related statistics, and a set of constitutive parameters for predicting stress-strain curves. This table can incorporate the relevant test information, ensuring that mechanical properties, model parameters, and associated statistics are readily available for each temperature and material.

4.2.4 Quality Control

In this example, the quality control process comprises three steps. The first step involves comparing the mechanical properties obtained with Paramaterial to those previously reported in the literature for the same dataset. The second step consists of checking for outliers using representative curves. Lastly, test reports are generated to document and store the information, enabling the review of individual test data and processing history.

The mechanical properties identified using Paramaterial are summarised in Table 4.6, and those presented in the literature [55] are summarised in Table 4.7. These values are then graphically compared in Figure 4.14.

Table 4.6: *Mechanical properties for the example dataset obtained using Paramaterial.*

Properties	20 °C	100 °C	150 °C	200 °C	250 °C	300 °C
E (GPa)	59.7 ±6.3	59.9 ±9.2	49.7 ±5.1	49.8 ±5.9	40.9 ±5.5	47.2 ±10.2
$\varepsilon_{\text{failure}}$ (%)	10.78 ±1.49	14.71 ±2.93	17.87 ±4.84	15.30 ±4.45	12.33 ±3.58	16.91 ±4.10
ε_{UTS} (%)	7.91 ±2.18	6.88 ±1.45	3.41 ±1.42	0.89 ±0.32	0.63 ±0.06	0.66 ±0.17
σ_{UTS} (MPa)	288.0 ±18.3	261.1 ±17.5	234.2 ±16.7	194.8 ±16.4	141.9 ±9.5	84.3 ±8.2
$\varepsilon_{\text{proof-0.2\%}}$ (%)	0.64 ±0.06	0.62 ±0.07	0.67 ±0.07	0.60 ±0.07	0.55 ±0.04	0.38 ±0.04
$\sigma_{\text{proof-0.2\%}}$ (MPa)	258.9 ±17.0	244.4 ±15.5	228.8 ±16.4	193.3 ±15.9	140.6 ±9.8	82.9 ±8.2

Table 4.7: *Mechanical properties for the example dataset reported in the literature.*

Properties	20 °C	100 °C	150 °C	200 °C	250 °C	300 °C
E (GPa)	60.5 ±5.40	59.1 ±5.48	51.0 ±4.47	50.2 ±5.86	40.3 ±3.79	39.7 ±9.06
$\varepsilon_{\text{failure}}$ (%)	10.8 ±2.0	14.7 ±3.0	17.9 ±5.0	15.0 ±4.0	11.7 ±3.0	15.2 ±4.0
ε_{UTS} (%)	8.23 ±1.0	6.91 ±1.0	3.47 ±1.0	0.924 ±0.3	0.686 ±0.05	0.724 ±0.2
σ_{UTS} (MPa)	288 ±17.8	261 ±17.5	234 ±16.7	195 ±14.8	142 ±9.54	84.1 ±8.02
$\varepsilon_{\text{proof-0.2\%}}$ (%)	0.632 ±0.0517	0.618 ±0.0594	0.647 ±0.0572	0.589 ±0.0729	0.551 ±0.0355	0.418 ±0.0476
$\sigma_{\text{proof-0.2\%}}$ (MPa)	259 ±16.9	245 ±15.5	229 ±16.5	194 ±14.4	141 ±9.89	82.9 ±7.96

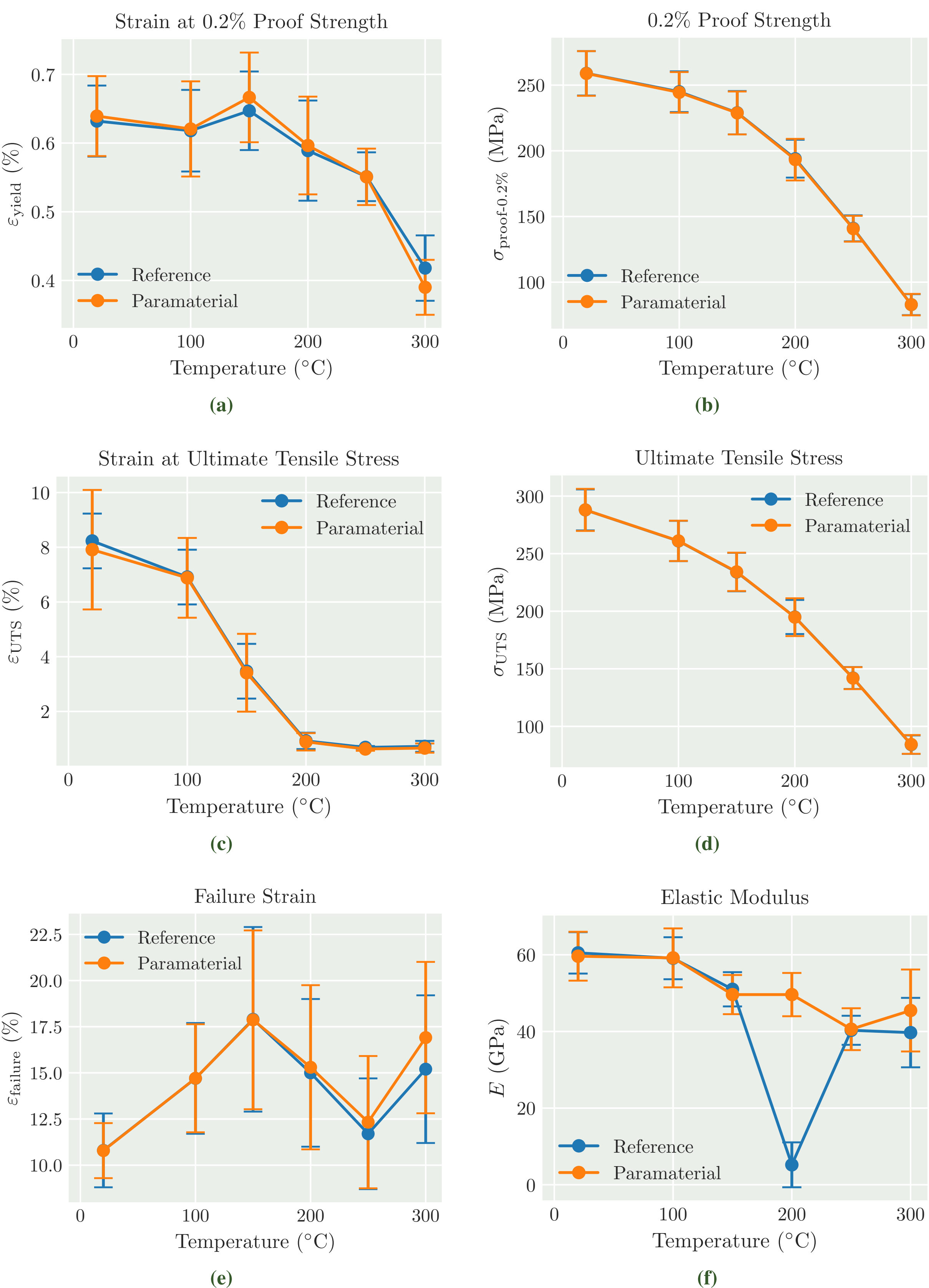


Figure 4.14: *Comparison of mechanical properties for the example dataset identified using Paramaterial against those reported in the literature [55].*

In general, the literature values closely align with the results obtained during the Paramaterial usage example. However, a significant exception is observed in the Young's modulus values, as seen in Figure 4.14f. The literature data exhibits a substantial and unrealistic decrease in Young's modulus at 200 °C, which is not present in the Paramaterial data. The Young's modulus values also differ at 300 °C, but to a lesser extent.

The discrepancies in the Young's modulus values are likely due to the screening step, wherein several tests were removed from the dataset (see Table 4.3). This rejected tests all at either 200 °C or 300 °C. Hence, it can be argued that the screening step enabled by Paramaterial led to a more accurate set of mechanical properties compared to those previously reported in the literature.

The next step in the quality control phase for this example is performing a check for outliers. The outlier check could have been performed at an earlier stage to remove them prior to further analysis. However, in this example the outliers are not removed from the dataset, but are simply noted.

The `ds_plot()` and `ds_subplots()` functions, defined in Listings 4.8 and 4.9, are useful for identifying outliers. They can be used to overlay the processed curves onto the representative curves to determine if any processed curves fall outside of one standard deviation[1] from the mean. This is demonstrated in Listing 4.21, which produces the grid of plots shown in Figure 4.15.

Upon examining Figure 4.15, it is evident that there are outliers according to the criterion of being more than one standard deviation from the mean. Outliers are present for Lot A at 100 °C, 200 °C, and 300 °C; for Lot C at 20 °C and 300 °C; and for Lot E at 150 °C. While it would be straightforward to use the screening PDF to mark these outliers, this is omitted in the example for brevity.

Listing 4.21: *Making a grid of subplots with representative curves to check for outliers.*

```
1  # Plot the processed curves on a grid of subplots
2  axs = ds_subplots(processed_ds)
3  
4  # Plot the representative curve for a row on all the plots in that row
5  for i, temp in enumerate(processed_ds.info_table['temperature'].unique()):
6      for j in range(axs.shape[1]):
7          ax = axs[i, j]
8          ds_plot(ds=repres_ds.subset({'temperature': temp}),
9                  ax=axs[i, j], plot_legend=False,
10                 fill_between=('up_std_Stress_MPa', 'down_std_Stress_MPa'))
```

The final step in the quality control phase of this example is the production of test reports that document the processing results for each test. Paramaterial provides a class, `TestReceipts`, for this purpose.

[1] Users can decide how tight to make this band. One standard deviation is chosen here for illustrative purposes.

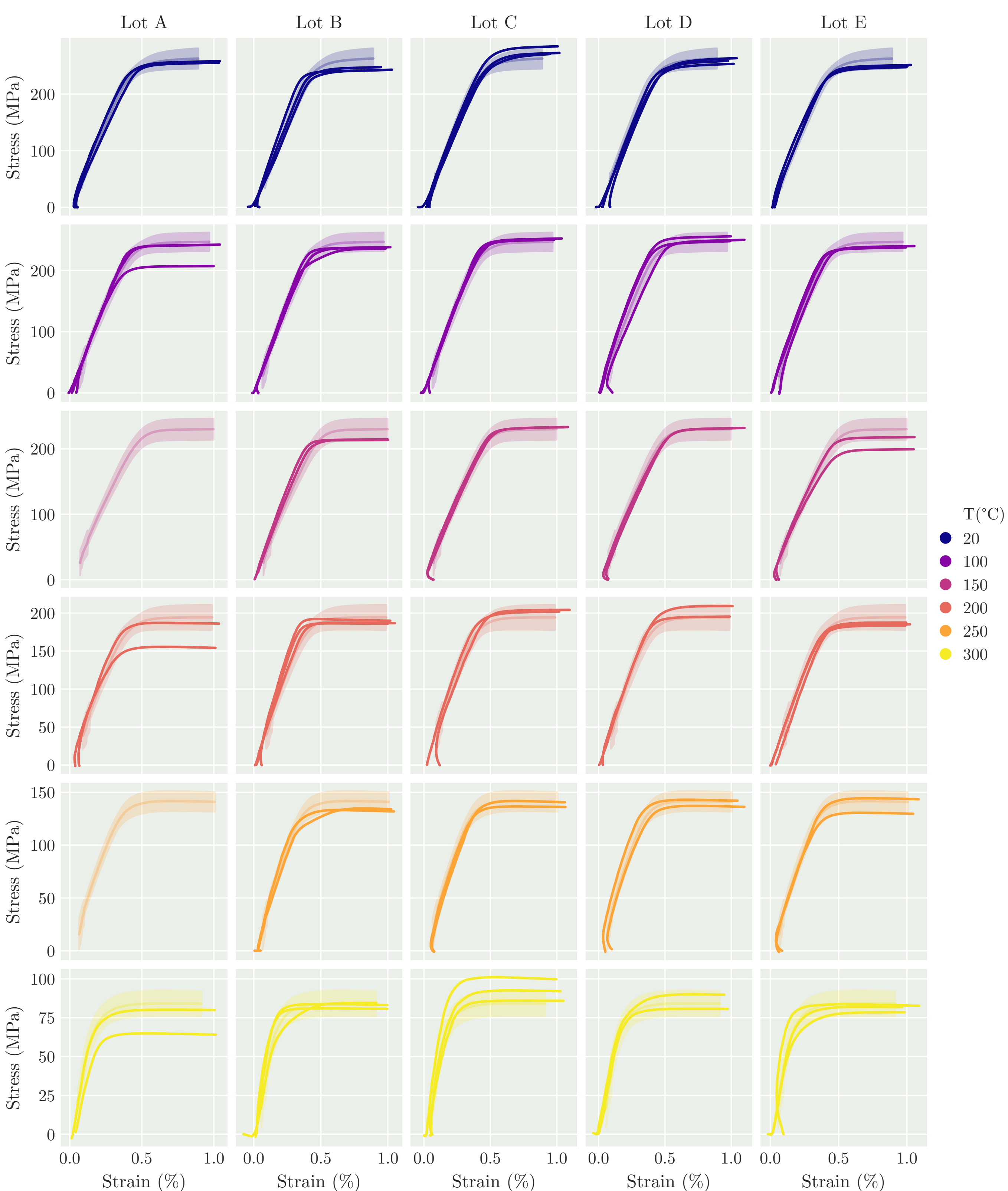


Figure 4.15: *Processed curves superimposed on the representative curves for each temperature grouping to identify outliers.*

The `TestReceipts` class makes use of the `Jinja2` templating library. This allows a Paramaterial user to define a template for their test reports using Latex. The template should contain placeholders, formatted as `\VAR{placeholder}`. The `TestReceipts.parse_placeholders()` function can be used to parse these placeholders, as demonstrated in Listing 4.22. This returns a list of all placeholders present in the template.

Listing 4.22: *Setting up the TestReceipts class and parsing the placeholders from a template.*

```
1  # Instantiate a TestReceipts object with the path to the template file
2  receipts = pam.TestReceipts(template_path='info/receipts_template.tex')
3
4  # Parse the placeholders from the template file
5  receipts.parse_placeholders()
```

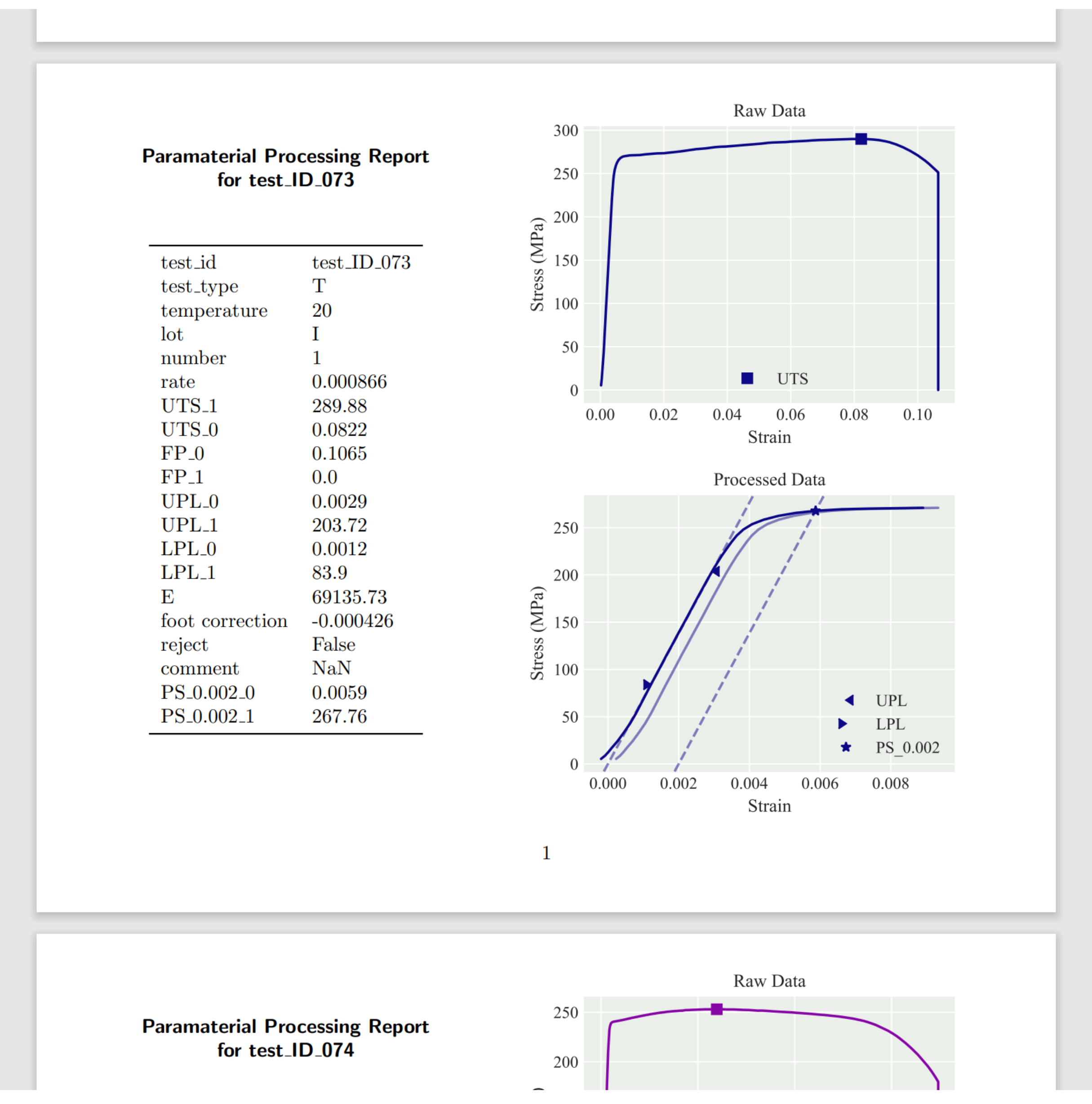
The `TestReceipts.generate_receipts()` function can then be employed, as shown in Listing 4.23, to create a PDF where each page is derived from the Latex template. For each test in the dataset, the placeholders in the template will be replaced using the information from the given test, before the test report is compiled. The resulting PDF will contain the test reports for all of the tests in the dataset.

Listing 4.23: *Generating a PDF of test reports for a dataset.*

```
1  # Define a dictionary of functions for replacing the placeholders
2  replace_dict = {'test_id': lambda di: di.test_id,
3                  'plot_raw': receipts_raw_plot,
4                  'plot_processed': receipts_processed_plot,
5                  'table_info': receipts_table}
6
7  # Generate the combined test receipts PDF
8  receipts.generate_receipts(ds=processed_ds,
9                             receipts_path='info/test_receipts.pdf',
10                            replace_dict=replace_dict)
```

The `replace_dict` argument of the `TestReceipts.generate_receipts()` function should be a dictionary containing a key for each placeholder in the template. The corresponding items must be either strings or functions. These functions can be used to generate plots or tables. Definitions for the `receipts_raw_plot()`, `receipts_processed_plot()`, and `receipts_table()` functions can be found in the Usage Example Jupyter Notebook, given in Appendix B.7.

A screenshot from the resulting PDF is given in Figure 4.16. The PDF includes a table and two plots on a single page for each test in the dataset. The table in the generated test report PDF displays all the metadata for the given test in the processed dataset. This includes the stress and strain values for the mechanical properties that were identified.



Paramaterial Processing Report for test_ID_073

test_id	test_ID_073
test_type	T
temperature	20
lot	I
number	1
rate	0.000866
UTS_1	289.88
UTS_0	0.0822
FP_0	0.1065
FP_1	0.0
UPL_0	0.0029
UPL_1	203.72
LPL_0	0.0012
LPL_1	83.9
E	69135.73
foot correction	-0.000426
reject	False
comment	NaN
PS_0.002_0	0.0059
PS_0.002_1	267.76

Paramaterial Processing Report for test_ID_074

Figure 4.16: *Screenshot from the tests receipts PDF.*

The plot on the upper right of the test report page in Figure 4.16 displays the raw stress-strain curve. The ultimate tensile strength (UTS) is marked on the raw curve with a square. The plot on the lower right of the page displays the processed stress-strain curve. The upper proportional limit (UPL) and lower proportional limit (LPL) are marked on the processed curve. The dashed line represents the elastic region. The shifted dashed line indicates the line used to identify the 0.2% proof strength, which is also marked on the processed curve. The faded curve represents the data prior to foot-correction.

This concludes the usage example. The full Jupyter Notebook for the example can be found in Appendix B.7. Detailed algorithms for some of the functions used in the example are presented in the next section.

4.3 Important Algorithms

Various important algorithms are presented in this section. These enable the automation of various data processing tasks, and have been implemented in functions provided by Paramaterial. Algorithms related to data processing are presented in Section 4.3.1, algorithms related to data aggregation are presented in Section 4.3.2, and algorithms related to fitting models are presented in Section 4.3.3.

4.3.1 Processing Algorithms

The `find_proportional_limits()` function, demonstrated in Listing 4.12, was implemented using Algorithm 1. Algorithm 1 identifies the lower proportional limit (LPL) and upper proportional limit (UPL) by fitting a line to a shifting window of data points, and was adapted from [57, 58].

Algorithm 1: *Finding the proportional limits of a stress-strain curve using linear regression.*

1. Given stress-strain data, $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$, start at a predefined stress, $\sigma_{preload}$.
2. Fit a line to the next few data points above the preload, using the following equations, and note the standard error, S_m.

$$\text{Slope:} \quad m = \frac{(\sum_{i=1}^{n} \varepsilon_i \sigma_i) - (\sum_{i=1}^{n} \varepsilon_i)(\sum_{i=1}^{n} \sigma_i)}{(\sum_{i=1}^{n} \varepsilon_i^2) - (\sum_{i=1}^{n} \varepsilon_i)^2},$$

$$\text{Standard deviation of } \varepsilon\text{-values:} \quad S_\varepsilon = \sqrt{\frac{\sum_{i=1}^{n} \varepsilon_i^2 - (\sum_{i=1}^{n} \varepsilon_i)^2}{n-1}},$$

$$\text{Standard deviation of } \sigma\text{-values:} \quad S_\sigma = \sqrt{\frac{\sum_{i=1}^{n} \sigma_i^2 - (\sum_{i=1}^{n} \sigma_i)^2}{n-1}},$$

$$\text{Covariance:} \quad S_{\varepsilon\sigma} = \frac{(\sum_{i=1}^{n} \varepsilon_i \sigma_i) - (\sum_{i=1}^{n} \varepsilon_i)(\sum_{i=1}^{n} \sigma_i)}{n-1},$$

$$\text{Correlation coefficient:} \quad r = \frac{S_{\varepsilon\sigma}}{S_\varepsilon S_\sigma},$$

$$\text{Standard error of slope:} \quad S_m = \frac{S_\sigma}{S_\varepsilon}\left(\frac{1-r^2}{n-2}\right)^{1/2}.$$

3. Iteratively extend the window of points and test the fit. If the error is less than the previous minimum, set the UPL as the data point at the end of the window. Do this until reaching the end of the data.
4. Start at the now-determined UPL and do the fitting procedure in the opposite direction to find the LPL.
5. Calculate Young's modulus as the gradient of a line between the UPL and LPL.

Following the identification of proportional limits, Algorithm 2 can be applied to perform foot correction. This is achieved by shifting the data until the line through the proportional limits intersects the origin. This algorithm was implemented in the `foot_correction()` function that was demonstrated in Listing 4.12.

Algorithm 2: *Applying foot correction.*

1. Start with given proportional limits (these can be identified using Algorithm 1).
2. Find the value of the x-intercept of a line through the proportional limits. Store this value as $\varepsilon_{\text{strain-shift}}$.
3. Subtract the value of the x-intercept from the strain-data, that is,

$$\varepsilon_{\text{corrected}} = \varepsilon_{\text{measured}} - \varepsilon_{\text{strain-shift}} \, .$$

   A line drawn through the proportional limits should now intercept the origin.

Once the proportional limits have been identified and foot correction applied, Algorithm 3 can be used to determine the proof stress. This algorithm was implemented in the `find_proof_stress()` function, demonstrated in Listing 4.16.

Algorithm 3: *Finding proof stress.*

1. Start with given proportional limits (these can be identified using Algorithm 1) and apply foot correction using Algorithm 2 if necessary.
2. Determine the equation of a line passing through the proportional limits and the origin.
3. Shift the line by the desired strain-offset. 0.1% or 0.2% strain are typical values.
4. Determine the intercept between the shifted line and the corrected stress-strain curve. Linearly interpolate between data points if necessary.
5. Store the stress value at the intercept as σ_{proof}

4.3.2 Aggregation Algorithms

A key step during aggregation is the identification of groupings within the data. Algorithm 4 is useful for finding combinations of key-value pairs for given categories to use as filters. After identifying groupings, Algorithm 5 can be used to generate aggregated data representing each group. Algorithm 4 and Algorithm 5 were used in the function `make_representative_data()`, that was demonstrated in Listing 4.18.

The data aggregation algorithms are used to generate representative curves. The representative curves usually show the mean of a group of tests, with a band for the standard deviation, or for

Algorithm 4: *Finding combinations of key-value pairs for given categories to use as filters.*

1. Start with n specified category keys, $\boldsymbol{k} = (k_1, k_2, ..., k_n)$, and find $\boldsymbol{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n)$, where $\boldsymbol{v}_i$ is a tuple of the the unique values in the category of k_i.

2. Make an empty tuple of filters, $\boldsymbol{F}_1 = ()$, then add filters using the first key:

   for v_j in $\boldsymbol{v}_1$ **do**
   Add the filter, $((k_1, v_j))$, to the tuple of filters, $\boldsymbol{F}_1$.
   end for

3. Update the filters using the remaining keys:

   for k_i and $\boldsymbol{v_i}$ in $\boldsymbol{k}$ and $\boldsymbol{V}$, with $i \neq 1$ **do**
   Make a new empty tuple of filters, $\boldsymbol{F}_2 = ()$.
   for f_k in $\boldsymbol{F}_1$ **do**
   for v_j in $\boldsymbol{v}_i$ **do**
   Add the pair, (k_i, v_j), to the tuple of pairs, f_k.
   Add the filter, f_k, to the tuple of filters, $\boldsymbol{F}_2$.
   end for
   end for
   $\boldsymbol{F}_1 = \boldsymbol{F}_2$.
   end for

4. The result is the set of filters, $\boldsymbol{F} = \boldsymbol{F}_1 = \boldsymbol{F}_2$, containing all the combinations of unique key-value pairs for the specified categories.

Algorithm 5: *Creating representative data.*

1. Start with identified groupings in the dataset (Algorithm 4 can be used to identify these).

2. Aggregate the time-series data in each grouping:

   2.1 Find minimum and maximum values of the strain vectors in the group.

   2.2 Make an artificial strain vector, $\varepsilon_{\text{mono}}$.

   2.3 Interpolate the stress vectors:

   for test **in** group **do**
   Interpolate the stress vector of the test with respect to $\varepsilon_{\text{mono}}$.
   Store the interpolated stress vector.
   end for

   2.4 Determine mean, standard deviation, maximum and minimum vectors for the interpolated stress vectors of the group.

3. Aggregate the metadata in each grouping:

   3.1 List the metadata categories to be aggregated.

   3.2 Determine mean, standard deviation, maximum and minimum values for each of the metadata categories in the grouping.

the maximum and minimum. The algorithms are also useful for producing plots of mechanical properties with error bars.

4.3.3 Model-Fitting Algorithms

The Ramberg-Osgood model, described by equations (2.8)-(2.15), and used in the model-fitting demonstration in Listing 4.19, is one of the built-in hardening models that can be imported from the `paramaterial.models` module. To implement these hardening models, it is necessary to derive an incremental form of the equations. This derivation, given in [59], results in Algorithm 6.

Algorithm 6: *Return-mapping algorithm.*

1. Start with known values at the current time-step, ε^n, ε_p^n, and α^n.
2. Increment the strain so that $\varepsilon^{n+1} = \varepsilon^n + \Delta\varepsilon^n$.
3. Calculate the trial stress as if the increment were elastic and check for yielding.

$$\sigma_{trial}^{n+1} \leftarrow E(\varepsilon^{n+1} - \varepsilon_p^n),$$
$$f_{trial}^{n+1} \leftarrow \left|\sigma_{trial}^{n+1}\right| - \sigma_f(\alpha^n),$$

If $f_{trial}^{n+1} \leq 0$, then the load step is elastic:

$\sigma^{n+1} = \sigma_{trial}^{n+1}$,

Exit the algorithm.

Else, if $f_{trial}^{n+1} > 0$, then the load step is elastic-plastic:

Continue at 4.

4. Calculate the stress, plastic strain, and accumulated plastic strain at the next time-step.

$$\Delta\gamma \leftarrow \text{solve}\left\{f_{trial}^{n+1} - \Delta\gamma E - \sigma_f(\alpha^{n+1}) + \sigma_f(\alpha^n) = 0\right\},$$
$$\sigma^{n+1} = \sigma_{trial}^{n+1} - \Delta\gamma \text{sign}(\sigma_{trial}^{n+1}),$$
$$\varepsilon_p^{n+1} = \varepsilon_p^n + \Delta\gamma \text{sign}(\sigma_{trial}^{n+1}),$$
$$\alpha^{n+1} = \alpha^n + \Delta\gamma,$$

Exit the algorithm.

The `ModelSet.fit()` function in Listing 4.19 was used to determine optimised material parameters for the `ramberg()` material model. The procedure used to determine the optimised parameters for the given material model is described in Algorithm 7.

Important algorithms used in the development of Paramaterial have been presented in this section. Additionally, some of the simpler procedures for correcting mechanical test data were previously addressed in the literature review[1], and several of the more complex correction methods are given in Appendix A.

[1] See Section 2.2

Algorithm 7: *Fitting a constitutive equation to stress-strain data.*

1. Define a function, $\sigma(\varepsilon, \boldsymbol{x}, \alpha)$, that gives stress, σ as a function of strain, ε, other known variables, $\boldsymbol{x}$, and material parameters to be optimised, $\boldsymbol{\alpha}$.
2. For a given test, extract the values of the known variables, $\boldsymbol{x}$, from the metadata.
3. Define some objective function that produces a scalar error for a given set of material parameters, $\boldsymbol{\alpha}_{\text{trial}}$. For example, the root mean squared error (RMSE) can be used:

$$RMSE = \sqrt{\frac{\sum_i^N \left[\sigma(\varepsilon_i^*, \boldsymbol{x}, \boldsymbol{\alpha}_{\text{trial}}) - \sigma_i^*\right]^2}{N}},$$

   where ε_i^* and σ_i^* are pairs of measured stress-strain data points, $\sigma(\varepsilon_i^*)$ is the stress predicted by the constitutive equation for a given ε_i^*, and N is the number of data points used during fitting.
4. Define a first guess and bounds for $\boldsymbol{\alpha}_{\text{trial}}$.
5. Start with the first guess, and iteratively test variations of $\boldsymbol{\alpha}_{\text{trial}}$ that are within the given bounds, until the objective function has been minimised to a specified tolerance.
6. Store the final optimised material parameters, $\boldsymbol{\alpha}_{\text{optimised}}$, along with the residual error.

The implementation and deployment of a package designed for repeatable processing of mechanical test data has been described in this chapter. Package details resembling a README were provided in Section 4.1. A usage example was then presented in Section 4.2. This illustrated how functionalities were implemented into compact front-end functions, hiding low-level details in the back-end, enabling users to accomplish tasks with minimal lines of code. Finally, some of the important algorithms used in the back-end implementation were presented in Section 4.3.

Additional implementation and deployment details are given in the appendices, including a description of the software architecture in Appendix B.1, a description of the contents and structure of the Paramaterial repository in Appendix B.2, descriptions of important classes in Appendix B.3, and descriptions of important functions in B.4. The full Jupyter Notebook for the usage example is also included in Appendix B.7.

5 Demonstration Case Studies

Well done is better than well said.

Benjamin Franklin

CASE *studies that demonstrate the usage of Paramaterial are presented in this chapter. There are four studies, given in Sections 5.1, 5.2, 5.3, and 5.4, respectively.*

The strategy for processing datasets[2] was applied in each case study to demonstrate the functionalities of Paramaterial. Each case study was undertaken with a specific research goal, or theme, in mind. The example datasets and corresponding research goals for the respective case studies are described as follows:

(CS1) *Variability in Material Properties of AA6061.* The inter-lot[3] and intra-lot variability of the properties of as-cast aluminium AA6061 are investigated in this study, using data from 76 uniaxial tension tests [60].

(CS2) *Comparing Uniaxial and Plane-Strain Tension Measurements.* The response of the same material, aluminium AA6061, under different loading conditions was compared in this study, using data from 20 uniaxial tension tests and 54 plain-strain tension tests [60].

(CS3) *Comparing As-Cast and Homogenised AA3104.* The study compared as-cast AA3104 with AA3104 homogenised at 560 °C and 580 °C, using 75 uniaxial compression tests at various temperatures and strain rates [61].

(CS4) *Plane-Strain Compression Test Specimen Geometry Effects.* Plain-strain compression test data [62] from 40 tests of aluminium AA3104 using two different specimen geometries, one smaller, and one larger, was compared in this study.

These studies are presented in the form of concise mini-reports to emphasize the efficiency with which such studies can be conducted using Paramaterial. The Jupyter Notebooks for the case studies are given in Appendices C.1, C.2, C.3, and C.4, respectively.

[2] See Section 3.2 for an outline of the strategy.

[3] A "lot" of material is an individual batch of that material from a given manufacturer. Properties of a nominally similar material can differ significantly across lots.

5.1 Variability in Material Properties of AA6061

The primary research goal of Case Study 1 (CS1) was to investigate variations in mechanical properties and material behaviour between different lots of the same material over a range of temperatures. The distribution of tests in the CS1 example dataset across lots and temperatures is given in Figure 5.1.

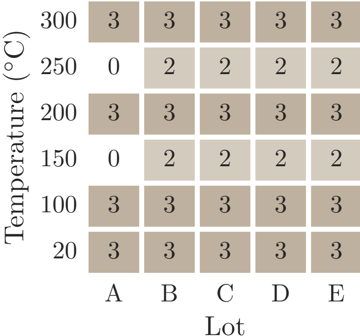


Figure 5.1: *Experimental matrix for CS1, showing distribution of tests across lots and temperatures.*

Various mechanical properties were determined from the CS1 example dataset using Paramaterial. These are presented in Table 5.1 with standard deviations, and plotted with error bands bounding maximum and minimum data in Figure 5.2.

Table 5.1: *Mechanical properties and standard deviations extracted from the CS1 example dataset.*

T (°C)	E (GPa)	$\sigma_{\text{proof-0.2\%}}$ (MPa)	σ_{UTS} (MPa)	ε_{UTS} (%)	$\varepsilon_{\text{failure}}$ (%)
20	59.9±6.1	252.6±11.2	282.1±12.0	8.2±1.3	10.9±1.6
100	62.2±8.8	238.7±11.2	256.0±15.1	6.6±1.3	15.4±2.5
150	50.7±4.8	220.2±12.1	226.7±14.2	3.8±0.5	20.1±4.2
200	51.8±4.6	188.4±13.2	189.6±13.6	0.9±0.3	16.7±3.7
250	40.1±5.9	136.4±5.5	137.7±4.9	0.6±0.1	14.1±2.5
300	48.3±10.1	82.3±8.3	83.8±8.4	0.7±0.2	17.6±3.8

The raw, processed, representative and fitted data for the CS1 example dataset is plotted in Figure 5.3. The curves are coloured by lot. Where multiple processed curves exist for a given lot and temperature, representative curves have been generated. The error bands of the representative curves show the maximum and minimum bounds at each data point. In Figure 5.3, the overall variability in material behaviour appears to be largest at 300 °C. Additionally, raw curves exhibit significant hardening at the lowest temperatures, while, conversely, the fracture drop-off evident in the tail-end of the raw data curves in Figure 5.3 appears to become more gradual with temperature.

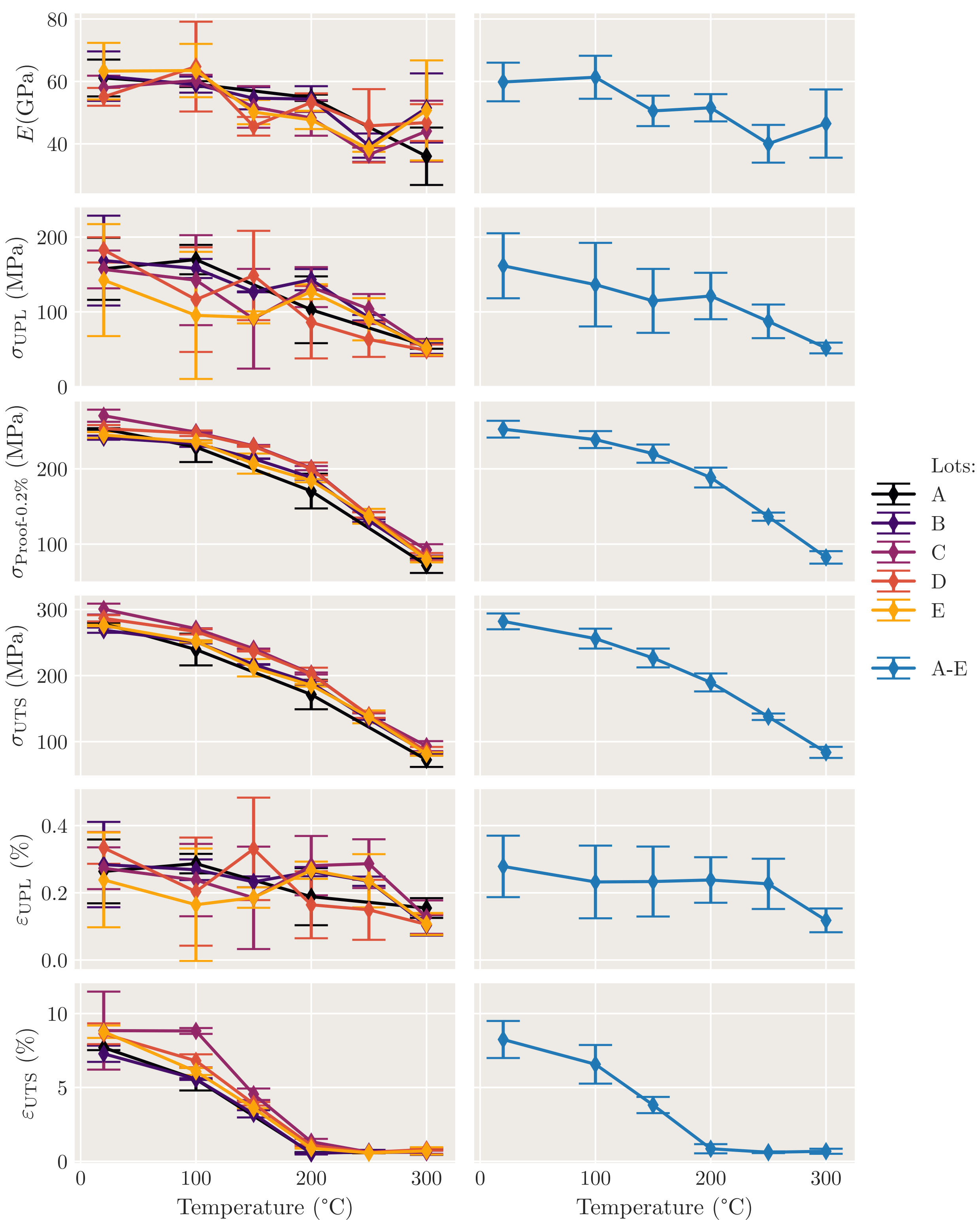


Figure 5.2: *Mechanical properties for lots A-E of CS1 example dataset plotted against temperature. Average values with standard deviations for each lot are shown left, and average values with standard deviations for the entire set are shown right.*

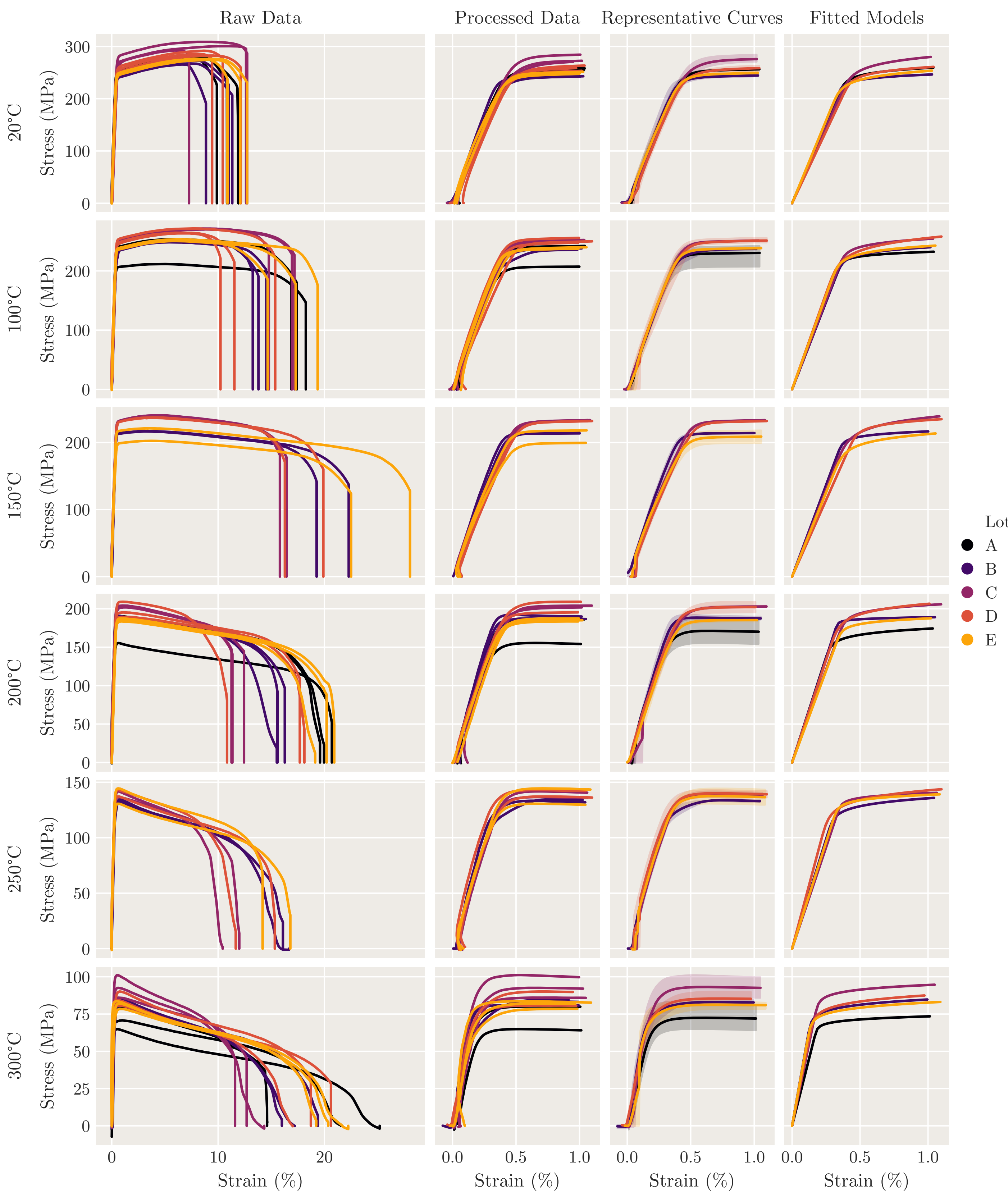


Figure 5.3: *Raw data, trimmed data, representative curves, and fitted models (from left to right, respectively) for the CS1 example dataset.*

Large error bars are evident in Figure 5.2 for the Young's modulus, E, stress at the upper proportional limit, σ_{UPL}, and strain at the upper proportional limit, ε_{UPL}. These three properties are related, as they were determined during the procedure outlined in Algorithm 1. These large error bars indicate that the procedure can be inaccurate for individual tests, implying that the results only become meaningful when a large enough dataset is used and the values are averaged.

The error bars are comparatively smaller for the values of the Proof Stress, $\sigma_{\text{proof-0.2\%}}$, the ultimate tensile stress (UTS), σ_{UTS}, and the strain at the UTS, ε_{UTS}. It is evident that some variation exists between the results from different lots for these properties. However, the inter-lot variation, indicated by the varying heights of individual curves, does not seem to be substantially larger than the intra-lot variation.

The plots on the right side of Figure 5.2 are more useful for identifying any temperature-related trends. The various strength measures, σ_{UPL}, $\sigma_{\text{proof-0.2\%}}$, and σ_{UTS}, all clearly decreased with temperature. The ductility measures, ε_{UPL} and ε_{UTS}, also appear to decrease with temperature. In fact, ε_{UTS} shows a marked drop for low strains, indicating that the material transitions from ductile yielding with hardening at lower temperatures to more brittle yielding at higher temperatures.

The Ramberg-Osgood model, described by equations (2.8)-(2.15), was fit to the CS1 example dataset using Paramaterial. The resulting fitted curves are shown alongside the raw data, processed data, and representative curves in Figure 5.3. Note that the model was fit to data for each temperature individually, and not to the whole set monolithically.

Optimised values for the hardening coefficient, K, and the strain-hardening exponent, n, and associated fitting errors, are given in Table 5.2. During fitting, the values for E and σ_Y were treated as known variables and taken as the previously identified mechanical properties, E and σ_{UPL}.

Table 5.2: *Model parameters for CS1. K and n where determined by optimisation, while E and σ_Y were determined during processing.*

T (°C)	E (GPa)	σ_Y (MPa)	K (MPa)	n	RMSE (MPa)
20	59.9±6.1	161.6±15.0	104.2±21.8	0.1±0.02	16.9±10.0
100	62.2±8.8	136.4±30.5	113.9±38.6	0.09±0.02	26.9±27.7
150	50.7±4.8	114.7±28.0	115.8±30.6	0.08±0.02	30.2±19.4
200	51.8±4.6	118.4±23.4	76.9±29.3	0.08±0.02	18.9±12.9
250	40.1±5.9	87.3±17.2	54.0±19.6	0.1±0.03	18.1±10.5
300	48.3±10.1	51.7±2.2	33.6±8.9	0.11±0.02	7.2±8.8

The optimised values for K, as fitted to the CS1 example dataset, have been plotted in Figure 5.4. The hardening coefficient, K, can be interpreted as representing the amount of available hardening. Under this interpretation, a lower value for K would suggest a more brittle material, and a higher

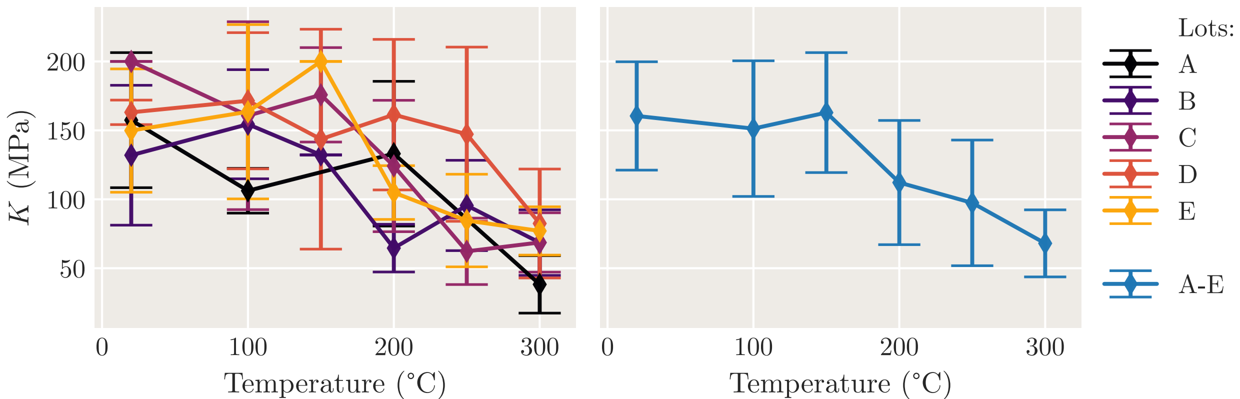


Figure 5.4: *Hardening coefficient values for the Ramberg-Osgood model fitted to the CS1 example data.*

value would suggest a more ductile material. While the error bars in Figure 5.4 are large, there does appear to be a decrease in K with temperature, that is, a decrease in "available hardening", for this material.

Note that while the data in this case study was trimmed to 1% strain, Paramaterial can also be used to analyse this data at higher strains. An example is provided at the end of Appendix C.1.

In this case study, results from processing 76 uniaxial tension tests have been presented. Mechanical properties were extracted and briefly analysed, and the raw data, processed data, representative curves, and fitted models were presented.

Key takeaways from this case study are as follows:

CS1.1 Both the inter-lot and intra-lot variation in material behaviour appear significant enough to be of consideration.

CS1.2 The procedure used to identify E, σ_{UPL}, and $\varepsilon_{\mathrm{UPL}}$ is prone to producing inaccurate results.

CS1.3 The mechanical properties measuring strength of the material show a clear decrease with temperature.

CS1.4 The strain at the UTS is high at lower temperatures, but decreases significantly at higher temperatures. This indicates that the material exhibits strain-hardening at lower temperatures, but not at higher temperatures.

CS1.5 The optimised values of the hardening coefficient, K, for the fitted Ramberg-Osgood model appear to decrease with temperature. This can be interpreted as showing that there is less "available hardening" for the material at higher temperatures.

5.2 Comparing Plane-Strain and Uniaxial Tension

The aim of Case Study 2 (CS2) was to compare the mechanical response of the same material under different loading conditions. The uniaxial and plane-strain tension test data covered 4 lots of AA6061 at 6 temperatures between 20 °C and 300 °C. This distribution of tests across these categories is displayed in the experimental matrix in Figure 5.5.

Temperature (°C)	Uniaxial F	Uniaxial G	Uniaxial H	Uniaxial I	Plane-Strain F	Plane-Strain G	Plane-Strain H	Plane-Strain I
300	1	1	1	1	3	3	3	3
250	1	1	1	1	3	3	3	3
200	1	1	1	1	0	3	3	3
150	1	1	1	1	0	3	3	3
20	1	1	1	1	3	3	3	3

Figure 5.5: *Experimental matrix showing the distribution of tests across lots, temperatures and test types.*

Plots of the raw stress-strain data for both test types are displayed in Figure 5.6, where it is evident that the maximum strains reached in the plane-strain tension tests (in the region of 18% to 50%) are significantly larger than those of the uniaxial tests (around 8% to 18% max strain).

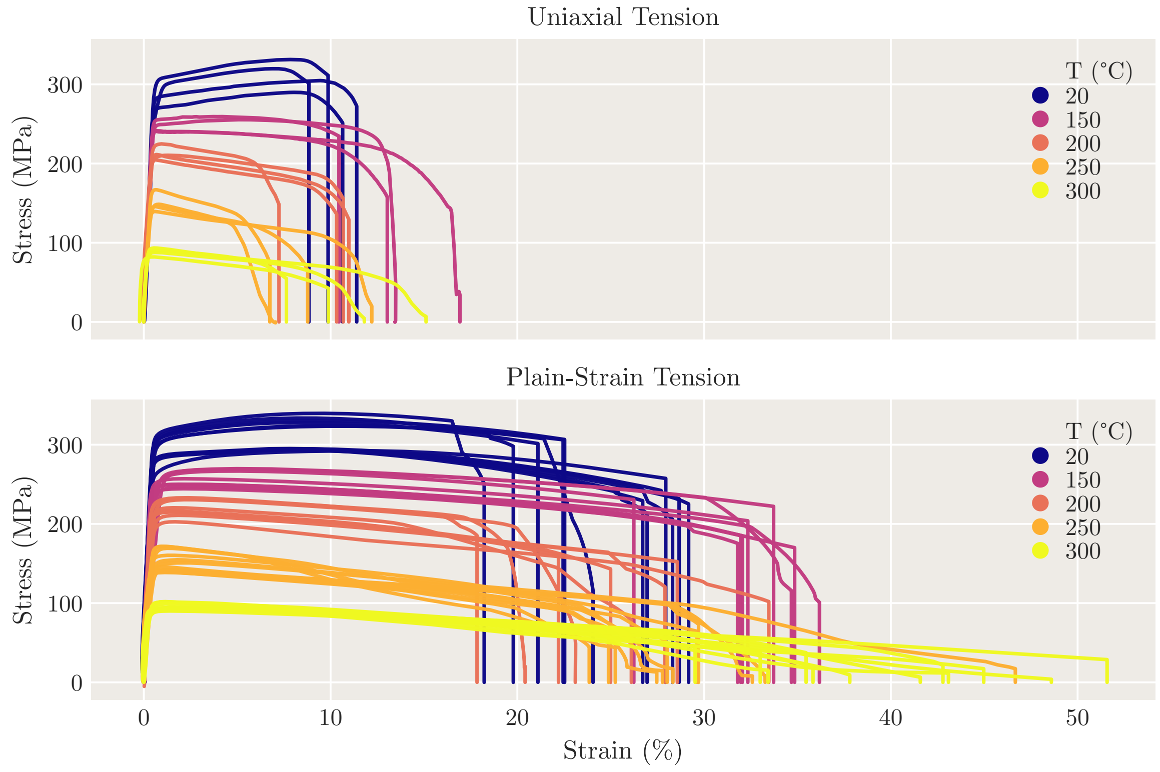


Figure 5.6: *Plots of the unprocessed uniaxial tension and plain-strain tension stress-strain data.*

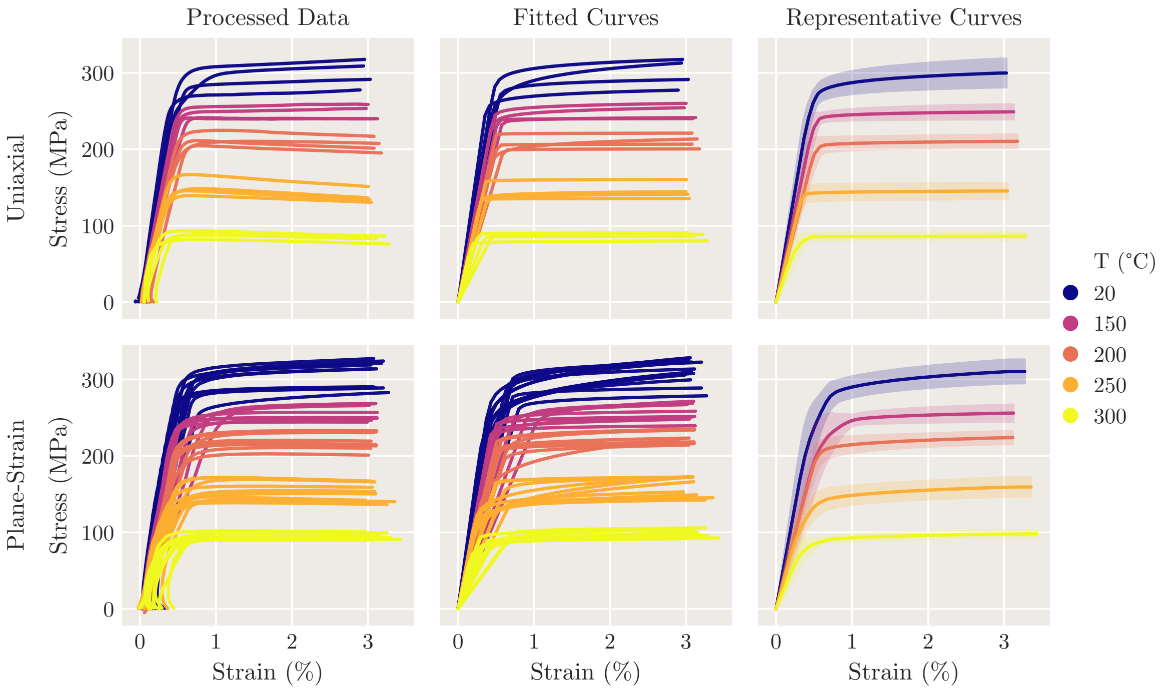


Figure 5.7: *Processed, fitted, and representative curves for the CS2 example dataset, with uniaxial tension tests in the top row and plane-strain tension tests in the bottom row.*

Table 5.3: *Model parameters for CS2. K and n where determined by optimisation, while E and σ_Y were determined during processing.*

T (°C)	Type	E (GPa)	σ_Y (MPa)	K (MPa)	n	RMSE (MPa)
20	PST	54.4±9.6	194.1±43.6	103.5±42.3	0.1±0.03	73.0±31.1
-	UT	58.9±8.0	169.5±56.9	123.4±69.5	0.06±0.01	9.4±6.3
150	PST	40.6±10.3	186.9±46.3	66.6±39.6	0.08±0.06	14.6±6.3
-	UT	47.8±5.8	163.3±45.2	83.6±51.3	0.02±0.01	12.8±9.0
200	PST	39.6±0.7	185.3±27.9	32.5±20.4	0.1±0.03	23.3±4.4
-	UT	39.6±4.7	157.1±41.3	51.8±39.8	0.02±0.01	62.2±47.2
250	PST	38.5±9.3	93.3±37.7	56.2±28.4	0.06±0.03	62.0±58.6
-	UT	42.4±4.9	84.5±38.6	59.9±36.8	0.01±0.0	35.4±29.4
300	PST	25.1±3.6	68.8±12.8	24.1±12.2	0.15±0.08	26.3±16.9
-	UT	31.2±12.6	68.5±9.4	17.5±5.9	0.01±0.0	41.8±38.5

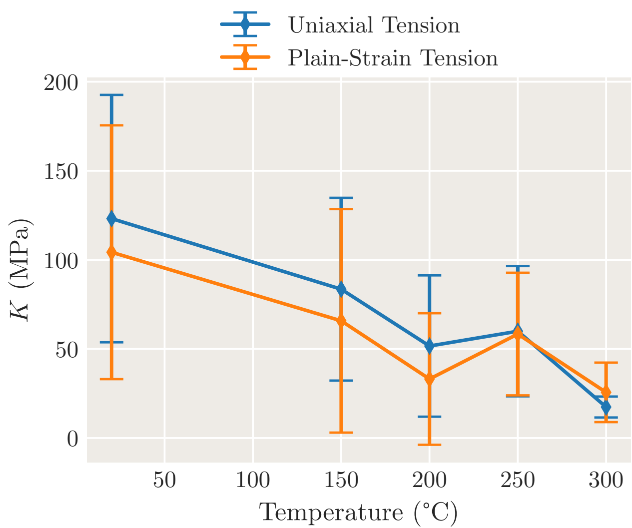


Figure 5.8: *Fitted values for the hardening coefficient, K, for the plane-strain and uniaxial test types.*

Table 5.4: *Mechanical properties for CS2.*

T (°C)	Type	E (GPa)	ε_{UPL} (%)	σ_{UPL} (MPa)	ε_{UTS} (%)	σ_{UTS} (MPa)	ε_{max} (%)
20	PST	54.4±9.6	0.4±0.2	194.1±43.6	9.0±0.8	311.9±21.0	24.7±4.1
-	UT	58.9±8.0	0.3±0.1	169.5±56.9	8.2±1.0	311.4±18.0	10.2±1.1
150	PST	40.6±10.3	0.5±0.2	186.9±46.3	3.2±1.7	255.9±11.2	32.6±1.4
-	UT	47.8±5.8	0.4±0.1	163.3±45.2	2.6±2.3	249.3±9.4	13.5±2.7
200	PST	39.6±0.7	0.5±0.1	185.3±27.9	1.8±0.4	219.5±11.2	25.0±4.8
-	UT	39.6±4.7	0.4±0.1	157.1±41.3	0.9±0.4	212.8±8.6	9.8±1.7
250	PST	38.5±9.3	0.3±0.2	93.3±37.7	1.3±0.3	152.4±13.9	29.8±4.2
-	UT	42.4±4.9	0.2±0.1	84.5±38.6	0.7±0.1	150.2±11.8	8.7±2.5
300	PST	25.1±3.6	0.3±0.1	68.8±12.8	1.9±0.6	95.0±3.0	39.8±5.1
-	UT	31.2±12.6	0.3±0.1	68.5±9.4	0.6±0.2	88.3±4.7	11.1±3.2

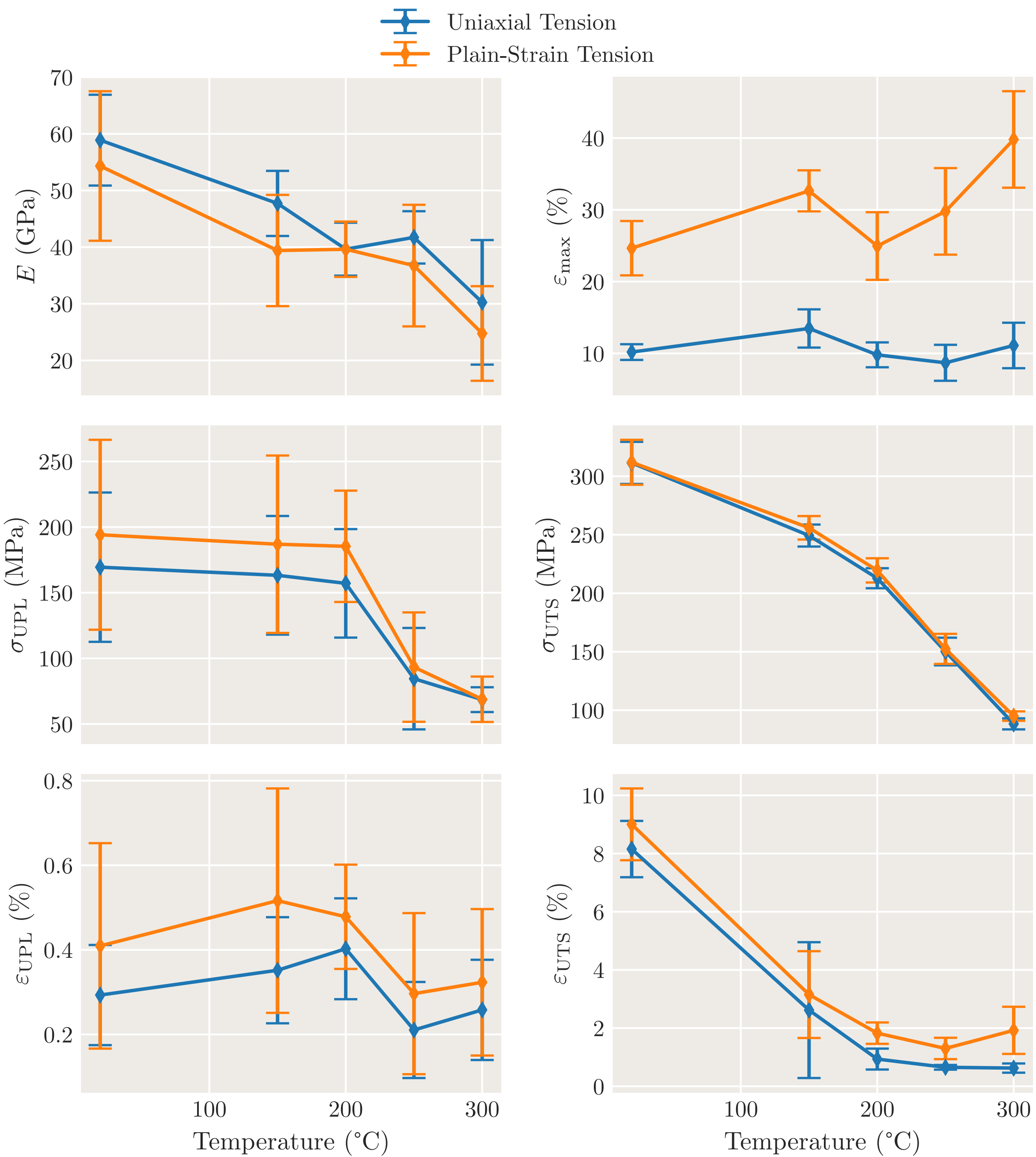


Figure 5.9: *Mechanical properties extracted from the uniaxial and plain-strain tension test data.*

Processed, fitted, and representative curves obtained using Paramaterial are displayed in Figure 5.7. Comparison of the processed uniaxial and plane-strain curves in the left column of the figure reveal no clear differences.

The fitted curves were produced using the Ramberg-Osgood model with E and σ_Y determined from the processing steps and the remaining parameters optimised for, as done in CS1. The fitted model parameters and errors are given in Table 5.3, and the hardening coefficient, K, is plotted for the two test types in Figure 5.8. The hardening coefficient is consistently higher for uniaxial tension than for plane-strain tension.

Extracted mechanical properties serve as a means to compare behaviour observed in different test types and at different temperature. The proportional limits, Young's modulus, UTS, and maximum strain have been determined from the processed curves using Paramaterial. Values for the mechanical properties and standard deviations are given in Table 5.4.

The mechanical properties are also plotted against temperature in Figure 5.9, where the error bars show standard deviations. The Young's modulus, E, maintains a consistently higher value for uniaxial tests. This is unexpected, and warrants further investigation of the methods used to produce this example data. Additionally, the maximum strain exhibits a considerably higher value for plane-strain tests (around 20% higher than for uniaxial tests), as previously observed in Figure 5.6. The ultimate tensile strength (UTS) results are similar (corresponding values are within about 5 MPa), suggesting that this metric can be compared across datasets from both test types. However, the strains at UTS are consistently about 1% higher for the plane-strain tests.

The following key takeaways summarise the analysis given in this case study:

CS2.1 Significantly higher maximum strains (around 20% strain higher) are achieved in the plain-strain tension tests.

CS2.2 The stress-strain curves of the two test types are similar in shape at small strains, that is, below 3% strain.

CS2.3 Fitted values for the Ramberg-Osgood model hardening coefficient, K, suggest more strain-hardening occurs in the uniaxial tests.

CS2.4 The plane-strain tension test Young's modulus, E, results are lower than those of the uniaxial tests. This is unexpected, and should be investigated further.

CS2.5 The ultimate tensile strength results of the two test types are consistently similar to within 5 MPa.

5.3 Effect of Heat Treatment on As-Cast AA3104

Uniaxial compression test data [61] was used in Case Study 3 (CS3) to investigate the effect of heat treatment on the flow stress of aluminium alloy AA3104. Three different material states: as-cast (AC), homogenised at 560 °C (H560), and homogenised at 580 °C (H580) were tested. The distribution of tests across material states, rates and temperatures is displayed in Figure 5.10.

Temperature (°C)	AC		H560		H580	
Rate (/s)	1	10	1	10	1	10
300	0	2	0	5	0	5
330	0	2	0	3	0	4
360	0	2	2	4	0	2
400	2	2	2	2	3	4
450	2	2	3	2	2	3
500	2	1	2	2	2	1

Figure 5.10: *Experimental matrix for CS3.*

The true stress at 0.3 strain (0.3 flow stress) was obtained for all of the tests using Paramaterial. This strength measure is plotted in Figure 5.11. The strength of the homogenised materials is clearly lower than that of the as-cast material, and the difference between the two materials homogenised at different temperatures is negligible.

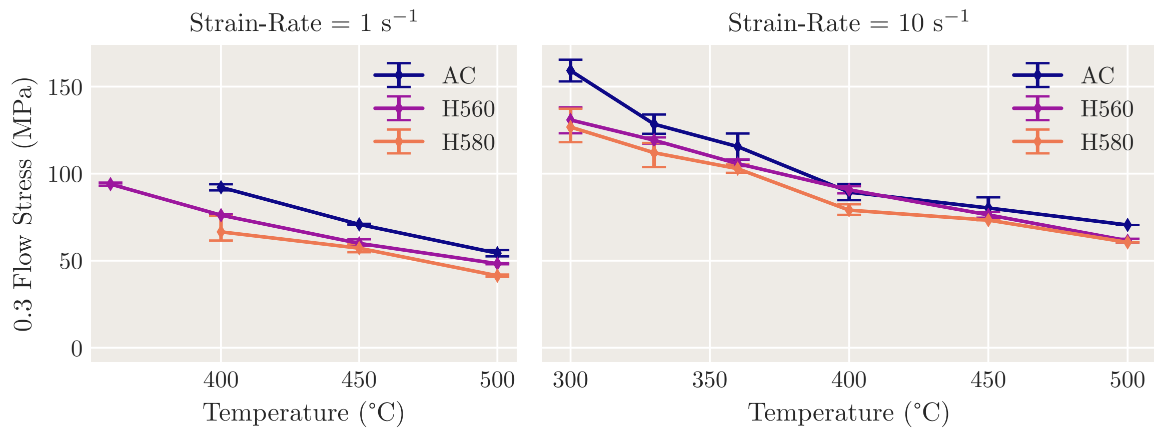


Figure 5.11: *Flow stress against temperature for AA3104 in the AC, H560, and H580 states.*

Figures 5.12a and 5.12b show the raw, processed, representative, and fitted data for tests coloured by nominal temperature. The curves are coloured by temperature. Note that the measured and nominal temperatures differ, in some cases by around 40 °C, as presented in Table 5.5.

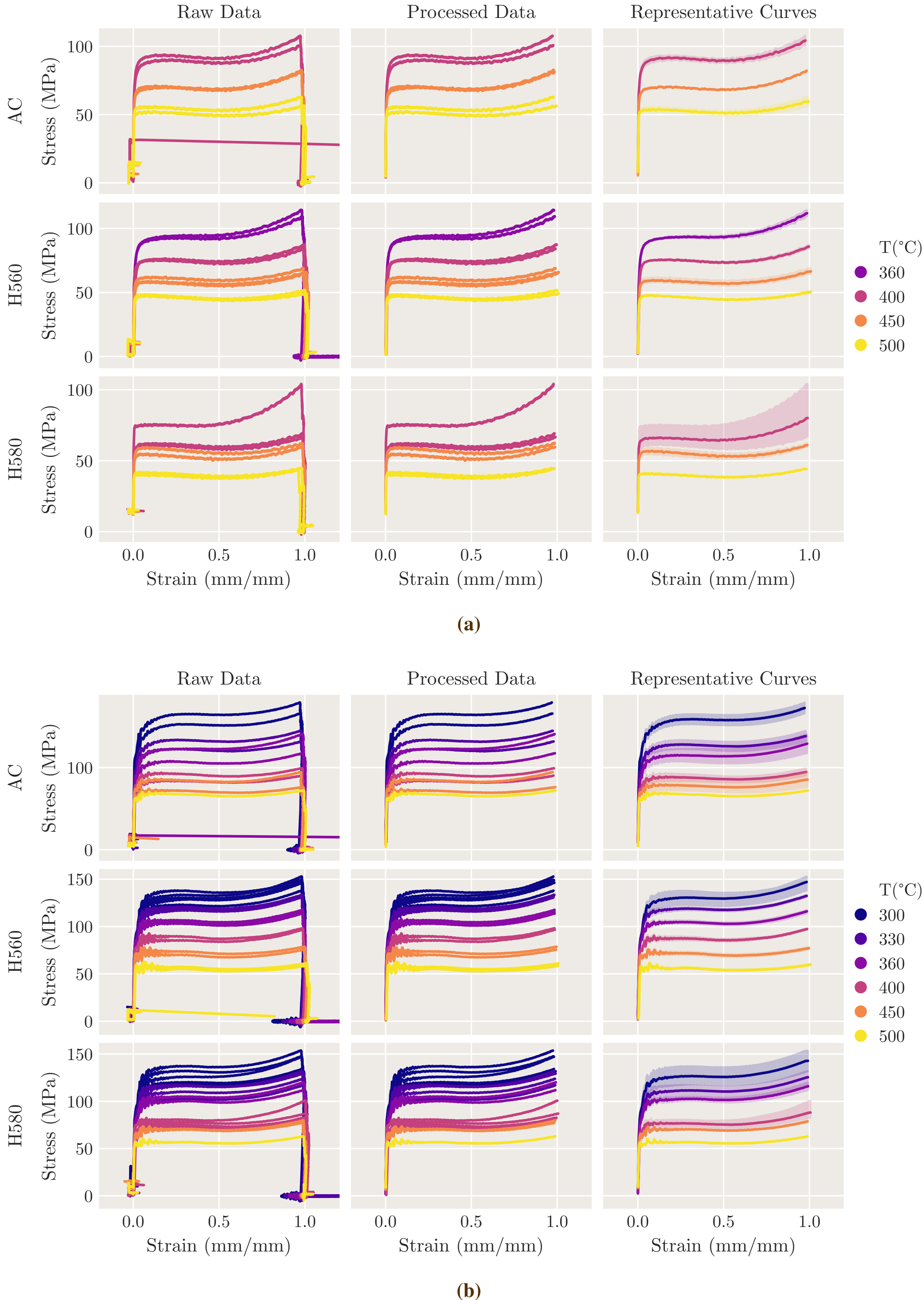


Figure 5.12: *Uniaxial compression test data for AA3104 at a strain-rate of, (a),* $1\,\mathrm{s}^{-1}$ *and, (b),* $10\,\mathrm{s}^{-1}$.

Table 5.5: *Flow-stress and Zener-Holloman values for CS3.*

$\dot{\varepsilon}$ (s^{-1})	$T_{nominal}$ (°C)	Material	$\sigma_{flow\text{-}0.3}$ (MPa)	$T_{flow\text{-}0.3}$ (°C)	$\ln Z$
1	360	H560	94	365	53.4
-	400	AC	92.1	405	48.1
-	-	H560	76.1	409	47.7
-	-	H580	66.5	447	43.7
-	450	AC	70.8	456	42.9
-	-	H560	59.9	458	42.7
-	-	H580	57.2	479	40.9
-	500	AC	54.3	506	38.8
-	-	H560	48.3	509	38.6
-	-	H580	41.3	534	36.7
10	300	AC	159	313	64
-	-	H560	131	314	63.9
-	-	H580	127	324	61.9
-	330	AC	128	359	56.2
-	-	H560	119	338	59.5
-	-	H580	112	355	56.9
-	360	AC	116	372	54.4
-	-	H560	106	368	55
-	-	H580	103	369	54.7
-	400	AC	89.4	436	46.7
-	-	H560	90.7	401	50.8
-	-	H580	79	434	47
-	450	AC	80.3	469	43.7
-	-	H560	76.3	457	45
-	-	H580	73.3	452	45.4
-	500	AC	70.5	500	41.2
-	-	H560	61.5	507	41
-	-	H580	60.5	500	41.3

Table 5.5 also contains values for the natural logarithm of the Zener-Holloman parameter, $\ln Z$. The associated flow-stress relation, see equation (2.22), was fit to the results for each material. The fitted regression lines are displayed in Figure 5.13a.

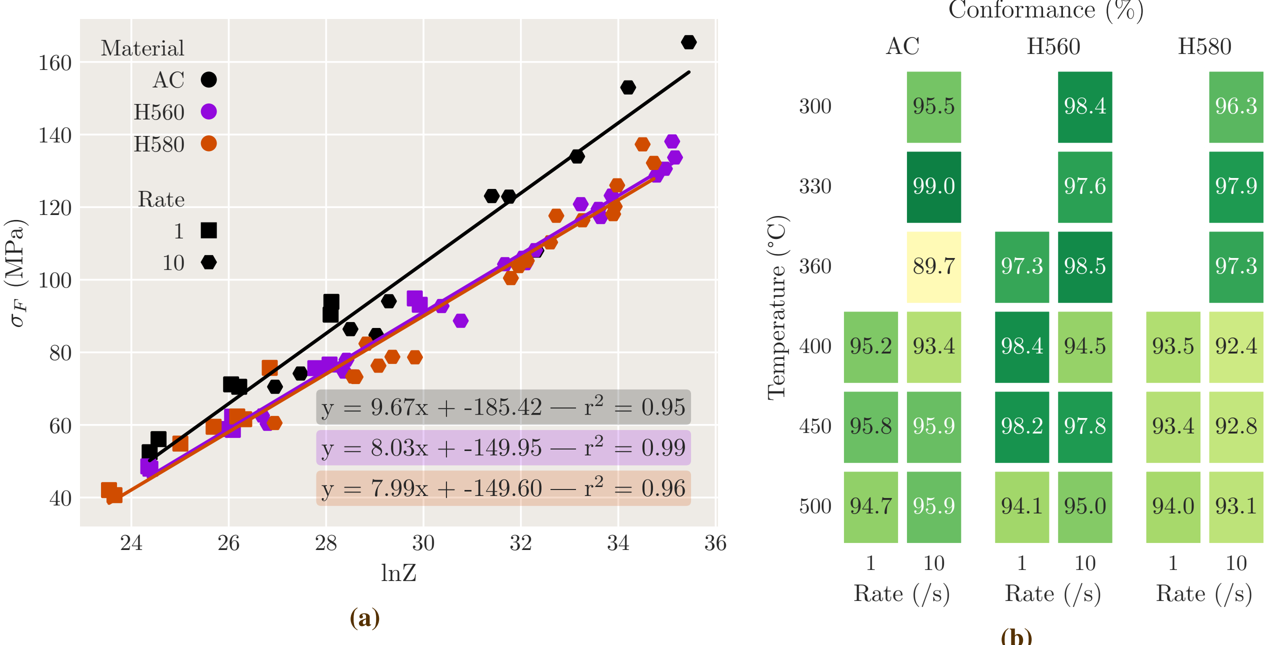


Figure 5.13: *(a), Zener-Holloman regression model fitted to uniaxial compression data for the different materials, and, (b), conformance of different temperature-rate subgroups to the respective fitted models.*

Fitting the model to the material-groups produced R^2 values of 0.95, 0.99 and 0.96, indicating that the fitted model describes these groups well. Notably, the regression lines fitted to the H560 and H580 material are approximately equivalent, with gradients of 8.03 and 7.99, respectively, and y-intercepts of -149.95 and -149.60, respectively.

The conformance values for subgroups of tests are displayed in Figure 5.13b. Each subgroup represents a group of tests performed at the same nominal temperature and strain rate for that material. The conformance value indicates how accurately the results for each subgroup are predicted by the regression line fitted to all of the tests for that material. This is calculated as

$$\text{Conformance (\%)} = \left(1 - \frac{\sum_i^N |\sigma_F(\ln Z_i^*) - (\sigma_F^*)_i|}{N \cdot \bar{\sigma}_F^*}\right) \cdot 100\,, \tag{5.1}$$

where $\sigma_F(\ln Z_i^*)$ is the predicted flow stress, $(\sigma_F^*)_i$ is the measured flow stress[1], N is the number of tests in the subgroup, and $\bar{\sigma}_F^*$ is the mean measured flow stress for that subgroup.

Subgroups with low conformance might require additional tests on the material at the temperature

[1]The measured flow stress for CS3 is given as $\sigma_{\text{flow-0.3}}$ (MPa) in Table 5.5.

and strain rate of the subgroup. Alternatively, low conformance of subgroups might indicate that the model is not capable of accurately capturing the material behaviour of these subgroups. In Figure 5.13b, the conformance of the subgroups ranges from 89.7% to 99.0%. The subgroup with the lowest conformance of 89.7% contains tests on the as-cast material that were performed at a nominal temperature of 300 °C and a nominal strain rate of $10\,s^{-1}$. The low conformance of this subgroup suggests that further tests in this category (per Figure 5.10, the subgroup currently contains two tests), would be beneficial.

The key takeaways for this case study are:

- **CS3.1** The $\sigma_{\text{flow-0.3}}$ results indicate that the flow strength of the homogenised materials is lower than that of the as-cast material.
- **CS3.2** The measured and nominal temperatures differ. In some cases this difference is in the region of 40 °C.
- **CS3.3** The regression lines for the homogenised materials fitted using the Zener-Holloman model are very similar, while the as-cast line clearly higher.
- **CS3.4** The as-cast tests at 360 °C and $10\,s^{-1}$, show a conformance to the Zener-Holloman model fitted to that material of less than 90%. This suggests that additional tests in this category might improve the dataset.

5.4 Plane-Strain Compression Geometry Effects

The PSC example dataset for this study was produced by a previous student at CME. The data was used to investigate the effect of specimen geometry on the flow stress of AA3104 transfer bar. Two specimen geometries were used: a smaller geometry (PSC) and a larger geometry (PSC*); see [62] for details. The distribution of tests across temperatures and rates is displayed in Figure 5.14, and the raw data and representative curves are plotted in Figure 5.15.

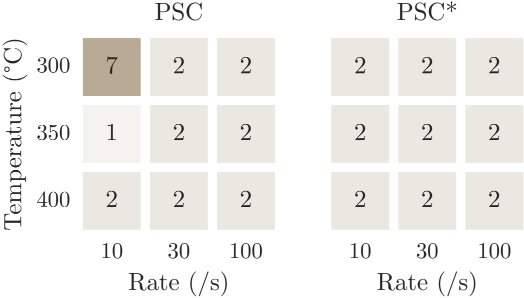


Figure 5.14: *Experimental matrix for CS4.*

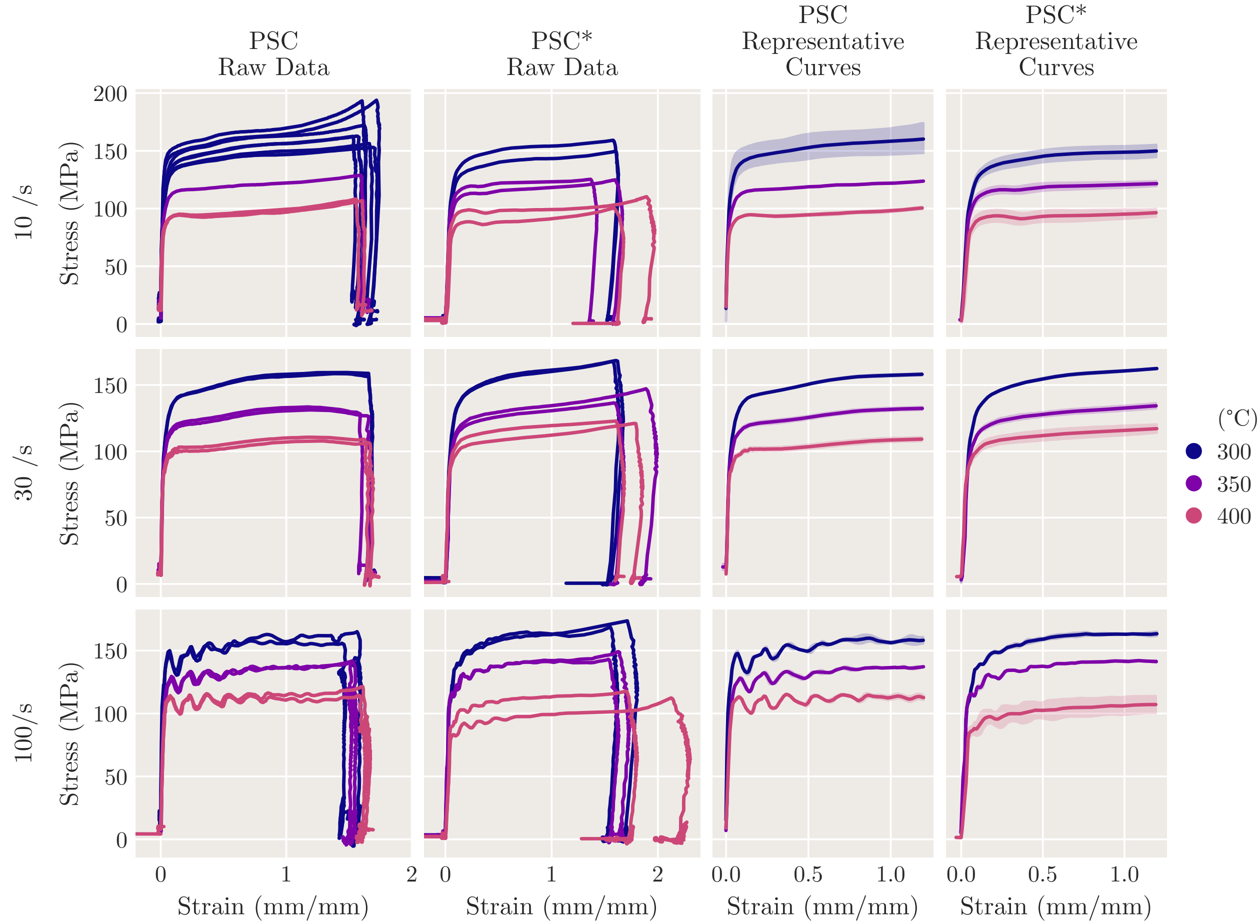


Figure 5.15: *Raw data and representative curves for AA3104 Transfer Bar from PSC and PSC* tests.*

Table 5.6: *Flow stress and Zener-Holloman values for CS4.*

$\dot{\varepsilon}$ (s^{-1})	$T_{nominal}$ (°C)	Test Type	$\sigma_{flow\text{-}0.8\text{-}0.9}$ (MPa)	$T_{flow\text{-}0.8\text{-}0.9}$ (°C)	ln Z
10	300	PSC	157	320	34.8
-	-	PSC*	148	328	34.3
-	350	PSC	121	371	32.2
-	-	PSC*	120	370	32.1
-	400	PSC	97.1	417	30.2
-	-	PSC*	94.5	415	30.3
30	300	PSC	157	349	34.7
-	-	PSC*	159	346	34.5
-	350	PSC	131	389	32.9
-	-	PSC*	131	396	32.4
-	400	PSC	108	428	31.2
-	-	PSC*	115	435	30.9
100	300	PSC	159	351	35.7
-	-	PSC*	163	350	35.2
-	350	PSC	136	387	33.6
-	-	PSC*	141	398	33.2
-	400	PSC	113	435	31.5
-	-	PSC*	106	452	31.2

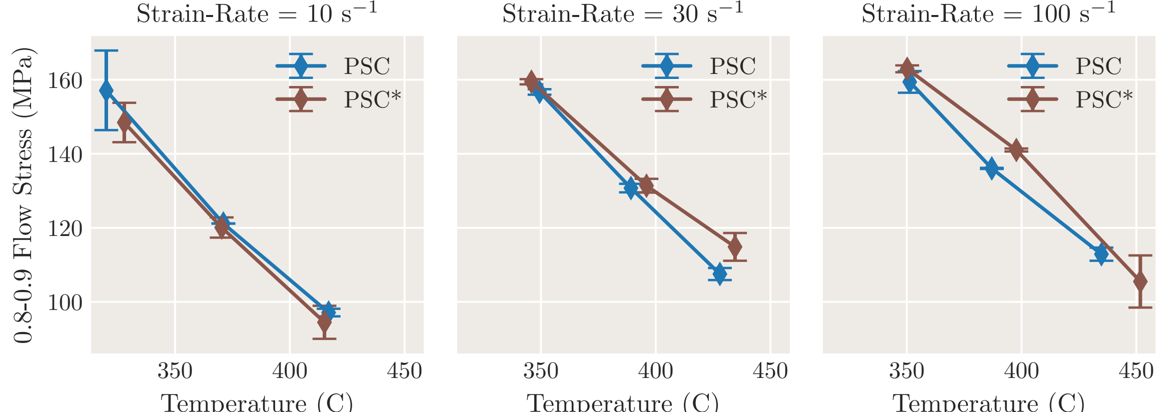


Figure 5.16: *0.3 flow stress values for AA3104 transfer bar, determined from plane-strain compression test data.*

The plotted curves in Figure 5.15 show similar shapes for the PSC and PSC* curves. The maximum strains achieved for tests on the larger specimens (PSC*) appear to be less consistent than for the smaller specimens (PSC). The maximum strains of the PSC tests are clustered within approximately 0.3 mm/mm, while the range of maximum strains for the PSC* tests is around 1 mm/mm. Additionally, oscillations are evident at higher rates. These oscillations are likely due to load-cell ringing.

The average true stress between 0.8 and 0.9 strain, that is, the 0.8-0.9 flow stress, was obtained for all of the tests. The average measured temperatures in this range were also determined. These values are presented in Table 5.6, in which the measured temperatures appear to be consistently around 50 °C higher than the nominal temperatures.

The flow stress results are plotted against measured temperatures in Figure 5.16, and flow stress results are fairly similar for both test types, with the maximum difference being around 10 MPa. The aforementioned temperature-shift compared to the nominal values appears less pronounced for the tests at a strain rate of 10 s^{-1}, as the lines for these tests are not shifted as far to the right as for the tests at higher rates.

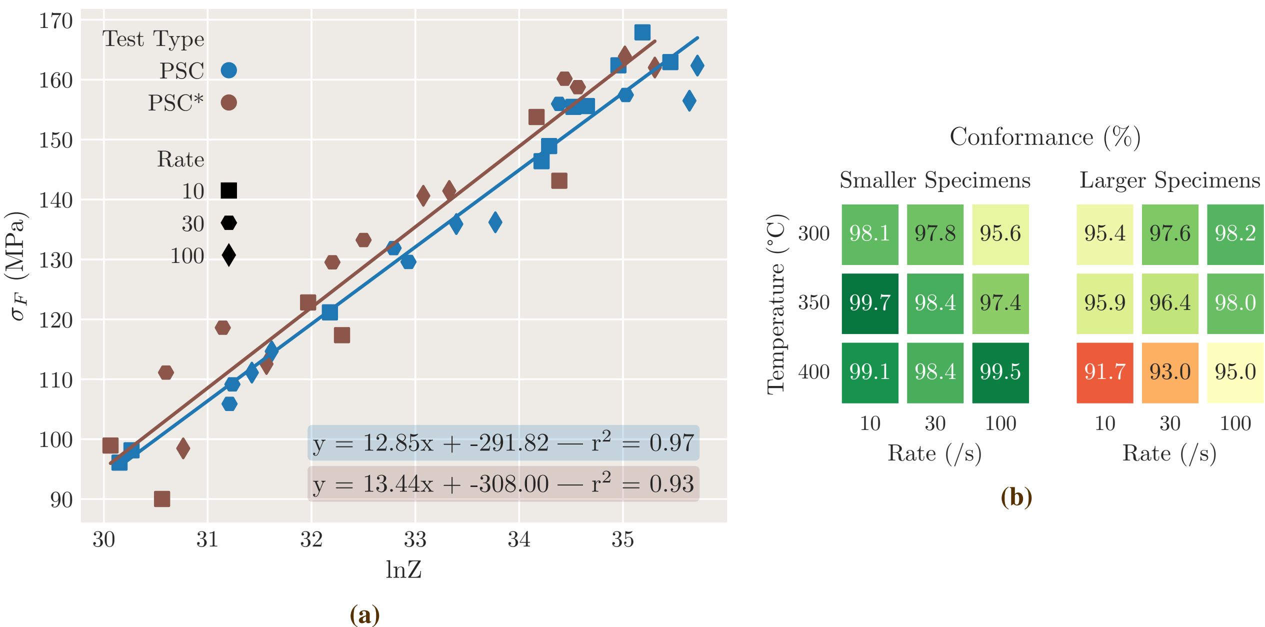


Figure 5.17: *Zener-Holloman regression model fitted to uniaxial compression data grouped by material state., (a), and conformance to model of different groupings, (b).*

The Zener-Holloman model was fit to the flow stress results for each of the two specimen types. The fitted regression lines are shown in Figure 5.17a, where it can be seen that the PSC* model consistently predicts a higher flow stress, σ_F, than that predicted by the PSC model. This difference is fairly small, though, and doesn't exceed 5 MPa. However the R^2 value is significantly lower for the PSC* model (0.93) than for the PSC model (0.97).

The conformance values for the different temperature-strain rate subgroups are shown in Figure 5.17b (see equation (5.1)). The conformance of the larger specimens (PSC*) is generally worse than for the smaller specimens (PSC). In particular, the poor conformance of the PSC* tests at $10\,s^{-1}$ and 400 °C, and $30\,s^{-1}$ and 400 °C, suggests that additional tests should be performed for these conditions.

The following are key takeaways from the preceding case study:

CS4.1 The shapes of the stress-strain curves are similar for both specimen geometries.

CS4.2 The measured flow stresses are similar for both specimen geometries (within 10 MPa), as are the flow stress values predicted by the fitted Zener-Holloman model (within 5 MPa).

CS4.3 There are significant differences between the measured temperatures and the nominal temperatures for both geometries. This is more pronounced at higher rates, and in many cases the difference is as large as 50 °C.

CS4.4 The low conformance to the Zener-Holloman model of results for larger specimens (PSC*) tested at $10\,s^{-1}$/400 °C, and at $30\,s^{-1}$/400 °C, indicate that further tests in these categories might be required.

In this chapter, four case studies demonstrating the functionalities of Paramaterial have been presented. The example datasets were processed in each case study using the strategy outlined in Section 3.2. The functionalities of Paramaterial are further discussed in the next chapter (Chapter 6), as are the qualities of the now feature-enriched example datasets.

6 Discussion

Study the past if you would define the future.

Confucius

THIS *chapter is divided into two sections. Section 6.1 is dedicated to a discussion of the functionalities of Paramaterial, and Section 6.2 contains an assessment of the qualities and features of the example datasets used in Chapter 5.*

6.1 Discussion of Functionalities

The specific purpose of each functionality is to assist the user in performing a particular task related to the processing of mechanical test data. The effectiveness of each functionality is discussed, and possible improvements are noted.

The functionalities are listed as follows, where items labelled ⚙ describe specific implementation details, 🔧 signifies possible improvements or features to be added, 📘 introduces details from the case study demonstrations, and 💬 heralds miscellaneous comments:

(F1) *Experimental Matrix.* The experimental matrix shows the distribution of tests across metadata features, which is useful for identifying groupings in the datasets prior to analysis.

- ⚙ The function `Paramaterial.experimental_matrix()` is used to create a commonly used grid overview showing the distribution of tests across metadata categories. Using one line of code, the user can get an overview of groupings within the dataset
- 📘 This functionality was used to identify the groupings of: lots and temperatures in CS1; test types, temperatures, and lots in CS2; material states, temperatures, and rates in CS3; and test types, materials, and rates in CS4.
- 🔧 The 2-D matrix format is only suitable for relatively simple datasets.
- 💬 The tool is particularly useful for exploratory data analysis.

(F2) *Determination of mechanical properties.* Mechanical properties were determined from the datasets and used to identify relationships between groupings or investigate material behaviour. Various functions for automatic determination of specific mechanical properties have been implemented[1].

- Examples of Paramaterial functions for finding mechanical properties are `find_UTS`, `find_flow_stress_values()`, and `find_proportional_limits()`. Additionally, the `DataSet.apply()` method can be used to apply custom functions for determining a property of a test to the entire dataset.
- These were demonstrated by finding the Young's modulus and proof strength in CS1 and CS2, and finding the flow stress in CS3 and CS4.
- Automated procedures can produce inaccurate or nonsensical results. These functionalities should be used with the screening PDF, allowing for a manual check.
- Identifying mechanical properties is often the primary goal of a user when processing a mechanical test dataset, making this functionality important.

(F3) *Data Overview.* Paramaterial provides several plotting functions to aid the user with visualising the data during processing. This functionality can be used to visually check fitted models and processing results, or to identify trends and differences between the groups.

- The `dataset_plot()` function is useful for plotting all of the curves from a dataset on a single plot, and the `dataset_subplots()` function is useful for rapidly generating subplots to compare different groupings side by side.
- The data overview plotting functions were used in the case studies to show large amounts of data in a compact format. It was also useful to group the data overview plots by processing stage, so that the raw, processed, representative, and fitted data could be easily compared.
- The `fill_between` argument to the plotting functions allows the user to easily plot error bands.

(F4) *Trimming.* The trimming functionality is useful for removing leading and trailing data, and for trimming the data down to small-strain.

- This functionality is achieved by defining a custom trimming function and applying it to the dataset using `DataSet.apply()`.
- Trimming data is necessary before certain algorithms can be applied to determine mechanical properties or to successfully fit constitutive models.

[1] Additionally, A GUI was developed that allows for manually selecting points on curves that represent mechanical properties, see Appendix B.5.

- The trimming functionality could be made more user-friendly by creating an in-built function.

(F5) *Corrections.* Various corrections functions have been implemented, for example, foot-correction and friction correction.

- The `correct_foot()` function was used to apply foot correction in the usage example and in CS1 and CS2.
- Stress and strain had already been calculated in the gathered example datasets, so extensive corrections were not necessary.

(F6) *Representative Curves.* Paramaterial provides functionality to calculate the representative curves from the test data. These curves represent the average behaviour of the material. Similarly, functionality is provided for averaging values stored in the metadata.

- The `make_representative_data()` function is used to generate representative curves from time-series data and the `make_representative_info()` is used to find statistics for metadata categories.
- Representative curves where generated and plotted for CS1, CS3, and CS4. These were particularly useful for visualising the variation in material properties in CS1, and for condensing the large numbers of repeated tests in CS3. This functionality can also be used to identify outliers by comparing individual tests to the representative curve from that group. However, care must be taken when discarding outliers from small groups of data. This is highlighted in CS3 Figure 5.12a, where the one test on the H580 material at 400 °C seems very far away from the other tests in the same group, but the mean curve still matches the expected trend when compared to the H560 curves.

(F7) *Fitting Models.* This functionality is useful for fitting models to data. This can be useful for predicting data points that weren't measured, for reproducing material behaviour in a simulation, or for analysing fitted model parameters that have inherent meaning.

- The `ModelSet` class provides methods for fitting a model to a dataset. This can be one of the built-in Paramaterial models, like `models.ramberg()`, or a user-defined model.
- This functionality was demonstrated by fitting the Ramberg-Osgood model to the data in CS1 and CS2, and by fitting the Zener-Holloman model to the metadata in CS3 and CS4.
- More complex material models than those used in the demonstrations

could also be fitted to identify trends in parameters with physical meaning.

(F8) *Conformance Matrix.* Paramaterial provides functionality to generate a matrix that shows the conformance of the test data to the constitutive equations. By assessing the conformance of data groupings, researchers can gain a better understanding of the applicability of the model and make informed decisions regarding further experiments or adjustments to the model.

- The `conformance_matrix()` function can be used to generate a conformance matrix for a model fit to the data or metadata of a dataset.
- The conformance matrix was used with the Zener-Holloman model that was fit to the identified mechanical properties metadata in CS3 and CS4. In CS4, the PSC* tests at 400 °C and $10\,s^{-1}$, and the AC material tests in CS3 at 360 °C and $10\,s^{-1}$ exhibit low conformance, and should be repeated to improve the quality of the datasets.

(F9) *Processing Notebooks.* The processing history can be captured in a Jupyter Notebook, which contains lines of code as well as graphics, and can be run from a browser.

- Paramaterial is designed to be used in a Jupyter Notebook. A Jupyter Notebook is a web-based interface that allows the user to write and execute code in a Python kernel. The user can run a series of code cells in the Jupyter Notebook to perform a task. The user can also write text in the Jupyter Notebook to document the task. The user can use the Jupyter Notebook to generate a report of the task.
- Processing reports for all cases studies are given in Appendix C. These can be adapted to process future datasets. These reports show that Paramaterial can be used to process and analyse datasets in a fully repeatable and traceable way.
- The Jupyter Notebook can be exported as a PDF document, to serve as a report. A user can also export the Jupyter Notebook as a Python script to reproduce the task. Additionally, the Jupyter Notebook can be exported in HTML, and added to the set of examples in the Paramaterial documentation.

(F10) *Interactive Screening PDFs.* The screening PDF allows the user to check the results of processing procedures on individual tests for a dataset.

- The screening PDF can be generated using the `make_screening_pdf()` function, and values entered into the PDF fields can be read with the `read_pdf_fields()` function.
- This functionality was demonstrated in the usage example in Section 4.2.
- The position of the elements on a page of the screening PDF is currently

hard-coded into the `make_screening_pdf()` function. The function could be developed to be more general.

- This functionality is useful when working with algorithms that are effective in most cases, but fail for some specific tests. The use of screening PDFs allows one to manually filter or correct errors from unseen automated processing.

(F11) *Generated test reports.* Paramaterial provides functionality for generating test reports from a LaTeX template. This allows the format of the test reports to be defined by the user. It is recommended that test reports be added to the processed dataset to improve traceability.

- The `TestReceipts` class provides methods for handling the templatting operations.
- This functionality was demonstrated in the usage example in Section 4.2.
- The interactive fields used in the screening PDFs functionality cannot currently be added to these LaTeX-generated test reports.
- The test reports can be setup to include information about the test, processing metadata, and model fitting results, as well as plots of raw data, processed data, and fitted models. The actual contents of a set of test reports should be guided by relevant standards and best practices guides. When generating test reports for a dataset after fitting models, it is useful to define an error threshold. If this threshold is exceeded for a given test, a flag can be raised, and a visible message printed on the test report. This can be used to pick up outliers or data with unexpected shapes.

Some of the functionalities of Paramaterial have been discussed in this section. The functionalities were used to process the gathered example datasets in Chapter 5, resulting in feature-enriched datasets with additional insights gained during processing. These processed datasets are discussed in the next section.

6.2 Discussion of Datasets

This section contains a comparison of the gathered and processed datasets. The contents and features of each dataset are summarised, and the quality of each dataset is analysed using a structured approach.

A set of qualitative terms of reference has been defined for discussing the quality of the processed datasets. These terms of reference, described in Table 6.1, provide a framework for assessing the quality of each dataset in terms of its repeatability, reproducibility, traceability, and general usefulness for further analysis.

Table 6.1: *Terms of reference for data quality comparison.*

Property	Description
Repeatability	The extent to which repeated experiments have produced consistent results.
Reproducibility	The extent to which an experiment can be replicated using different equipment.
Traceability	The ability to track the origins of data and how it has been processed.
Usefulness	The utility of the processed dataset for further analysis.

For the purposes of this discussion, a high quality dataset is one for which the properties described in Table 6.1 are satisfied. In the following, the data quality of the four different case studies is evaluated using these terms of reference:

Dataset 1: Uniaxial Tension Test Data

Number of tests: 100

Features:

- 9 material lots of AA6061
- Temperatures of 20 °C, 100 °C, 150 °C, 200 °C, 250 °C, and 300 °C
- Proportional limits, Young's moduli, proof strengths, and Ramberg-Osgood model fitted parameters

Quality Score: ●●●○

Property	Score	Motivation
Repeatability	●	Repeated tests are mostly very similar. While a manual screening step was performed after foot correction, this step is still repeatable as the screening results are recorded in the processing notebook
Reproducibility	●	The experimental procedure was outlined in detail in [60].
Traceability	◑	The published data didn't contain any force, displacement, time, or temperature measurements, only stress and strain measurements.
Usefulness	◑	This could be improved if the dataset contained tests over a range of strain-rates.

● = satisfied; ◑ = partially satisfied; ○ = not satisfied

Dataset 2: Plane-Strain Tension Data

Number of tests: 56

Features:

- 4 material lots of AA6061
- Temperatures of 20 °C, 150 °C, 200 °C, 250 °C, and 300 °C
- Proportional limits, Young's moduli, proof strengths, Ramberg-Osgood model fitted parameters

Quality Score: ●●◐○

Property	Score	Motivation
Repeatability	●	Repeated tests are mostly very similar, and multiple tests were performed for most categories.
Reproducibility	◑	The experimental procedure was outlined in detail in [60]. (Dataset 1 and Dataset 2 were published together online.) However, the reproducibility requirement is only partially satisfied as a Finite Element Analysis (FEA) was used to process the results prior to them being published, and the FEA was not reported on in detail.
Traceability	◑	The published data didn't contain any force, displacement, time, or temperature measurements, only stress and strain measurements.
Usefulness	◑	This could be improved if the dataset contained tests over a range of strain-rates. Plane-strain tension results are also less useful than uniaxial tension results, as constitutive models usually assume the data is from a uniaxial test.

● = satisfied; ◑ = partially satisfied; ○ = not satisfied

Dataset 3: Uniaxial Compression Test Data

Number of tests: 70

Features:

- 3 material states of AA3104
- Temperatures of 300 °C, 330 °C, 360 °C, 400 °C, 450 °C, and 500 °C
- Strain rates of 1 s^{-1} and 10 s^{-1}
- Flow stresses, Zener-Holloman model fitted parameters

Quality Score: ●●○○

Property	Score	Motivation
Repeatability	●	Tests in all categories were repeated, and repeated tests produced similar results.
Reproducibility	◑	There is very little reported information about the experimental procedures, but the Gleeble programming files that describe the experiment parameters for the machine are available for all of the tests.
Traceability	○	Experimental details were not reported.
Usefulness	◑	The dataset covers a range of materials, temperatures, and strain-rates, making it a useful practice dataset, even though its poor traceability makes it unsuitable for producing publishable research.

● = satisfied; ◑ = partially satisfied; ○ = not satisfied

Dataset 4: Plane-Strain Compression Test Data

Number of tests: 40

Features:

- 2 specimen geometries for AA3104 transfer bar at temperatures of 300 °C, 400 °C and 500 °C
- Strain rates of $10\,s^{-1}$, $30\,s^{-1}$, and $100\,s^{-1}$
- Flow stresses, Zener-Holloman model fitted parameters

Quality Score: ●●◐○

Property	Score	Motivation
Repeatability	○	Most tests were only repeated once, and results were often not similar.
Reproducibility	●	The experimental procedures are thoroughly detailed in [62].
Traceability	●	The experimental procedures are well detailed, and the Gleeble programming files are available for each test.
Usefulness	◑	The high traceability makes it possible to identify the causes behind trends observed in the data, even though there are relatively few tests in the dataset.

● = satisfied; ◑ = partially satisfied; ○ = not satisfied

In the preceding quality analysis, the four example datasets were assigned quality scores. The quality scores are summarised in Table 6.2. The uniaxial tension dataset was given the highest score, the plain-strain tension and compression datasets were tied with the second highest score,

Table 6.2: *Quality scores for the example datasets.*

	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Repeatability	●	●	●	○
Reproducibility	●	◑	◑	●
Traceability	◑	◑	○	●
Usefulness	◑	◑	◑	◑
Quality	○●●●	○◑●●	○○●●	○◑●●

● = satisfied; ◑ = partially satisfied; ○ = not satisfied

and the uniaxial compression dataset scored lowest. The uniaxial compression test dataset exhibited particularly poor traceability, and should only be used as a practice or example dataset.

A discussion of the functionalities of Paramaterial has been presented in this chapter, including their implementation and demonstration. A quality assessment of the example datasets was also presented, highlighting their features and limitations. The chapter has served not only to showcase the utility of Paramaterial for processing mechanical test data, but also to emphasise the importance of maintaining repeatability and traceability in these types of datasets.

7 Conclusions

The best way to predict the future is to create it.

Abraham Lincoln

CONCLUSIONS *from the present work and recommendations for future work are given in this chapter. A summary of the dissertation is given in Section 7.1, followed by a review of key outcomes in Section 7.2. Recommendations are presented in Section 7.3, a reflection on the objectives is provided in Section 7.4, and closing remarks are given in Section 7.5.*

7.1 Summary

The aim of this project was to develop an open-source software package for repeatable processing of mechanical test data. Relevant aspects of the work were reported in this dissertation as follows:

- Important literature was reviewed in Chapter 2.
- Software design considerations were presented in Chapter 3.
- The implementation and deployment of the package were detailed in Chapter 4.
- The utility of the package was demonstrated using case studies in Chapter 5.
- A discussion of the software's functionalities, as well as a comparison of the quality of the case study datasets, was included in Chapter 6.

The literature review, Chapter 2, covered theory on modelling material behaviour, and on obtaining data from mechanical tests. Four specific test types were covered: uniaxial tension tests, plane-strain tension tests, uniaxial compression tests, and plane-strain compression tests. These test types correspond to the test types of the four example datasets that were used in the case studies.

In Chapter 3, requirements for processing a mechanical test dataset in a repeatable way were identified based on the contents of the literature review. This information, along with preliminary investigations of the gathered example datasets, was used to outline a dataset processing strategy. The processing strategy consisted of four phases: data preparation, data processing, data aggregation, and quality control. This strategy was used to inform the software design requirements that were presented thereafter.

The developed open-source Python package, Paramaterial, was then introduced in Chapter 4. An overview of the deployed package was given with installation instructions, before a detailed usage example was presented. The usage example showed how Paramaterial can be used to complete the four phases of the data processing strategy. Important algorithms used in the implementation of automated processing procedures were also presented.

Comparison of mechanical properties extracted as part of the usage example with those reported in the literature for the same dataset revealed a significant difference. The literature results had included values for Young's modulus that were physically unlikely[1], while the values produced by Paramaterial were more realistic. The same algorithm was used to obtain both sets of results, so the difference was attributed to the manual screening functionality provided by Paramaterial.

The efficacy of Paramaterial for processing mechanical test datasets was then demonstrated in Chapter 5, by using the package to process four case study datasets. The results from the case studies were presented in the format of information-dense mini-reports. In these reports, the raw and processed stress-strain data, extracted mechanical properties, and fitted models were used to identify key analysis insights for the different datasets.

A discussion of the functionalities demonstrated in the usage example and leveraged in the case studies was presented next, in Chapter 6, with reference to specific function names, usage limitations, and miscellaneous comments. Additionally, an overview of the resulting feature-enriched datasets was provided, alongside a discussion of the quality of the different datasets. To aid in the discussion of quality, the datasets were assigned a quality score. This quality score was based on the following qualitative metrics: repeatability, reproducibility, traceability, and usefulness.

Jupyter Notebooks containing the code used for the usage example and case studies are included in the appendix. The code in these Notebooks can be run to repeat the analyses presented in this report. Thus, the developed software package, Paramaterial, has been shown to be an effective tool for repeatable processing and analysis of mechanical test data.

[1] See Figure 4.14f.

7.2 Key Outcomes

The development and usage of Paramaterial, an open-source data processing and analysis toolkit for mechanical engineering datasets, has been presented in this dissertation. Key outcomes for the project are as follows:

- **Dataset Processing Strategy:** Testing standards and best-practice guides were consulted during development of a dataset processing strategy. The result is a general guide for processing mechanical test datasets, with an emphasis on maintaining traceability and repeatability.
- **Software Package:** Various tools for applying the processing strategy were developed, wrapped into documented functions, and published as an open-source software package named Paramaterial. Paramaterial is a versatile and extensible Python package that streamlines the processing and analysis of experimental data, while maintaining repeatability and traceability. The package provides specific functions for finding mechanical properties, such as Young's modulus and yield strength. More generally-useful functions for ensuring repeatability and traceability are also provided, including functions for producing interactive screening PDFs, and for automated generation of test reports from a LaTeX template. The package was designed to be used in a Jupyter Notebook, which facilitates the production of repeatable processing reports that contain the code used to manipulate the data alongside graphics of the data at different stages. This approach ensures that the processing history is properly recorded, and provides examples that can be adapted to process new datasets.
- **Case Studies:** Paramaterial was used to apply the processing strategy in four case studies, demonstrating its efficacy for analysing mechanical test datasets in a repeatable and traceable way. Furthermore, these studies highlight how the functionalities of Paramaterial for producing visualisations, extracting mechanical properties, generating representative curves, and fitting constitutive models can be used to rapidly produce analysis reports with key insights about relationships between different features in the dataset.
- **Combined Example Dataset:** The gathered example datasets were processed and analysed in the case studies. Additionally, a quality analysis of each of the datasets was provided in the discussion. The resulting prepared, cleaned, processed, and feature-enriched datasets together form a combined example dataset that is in itself a significant outcome of this project, and provides a resource for further analysis and teaching purposes. This combined dataset consists of 100 uniaxial tensile tests on aluminium AA6061, 56 plane-strain tension tests on aluminium AA6061, 80 uniaxial compression tests on aluminium AA3104, and 40 plane-strain compression tests on aluminium AA3104.

7.3 Recommendations

The recommendations presented in this section are grouped into two categories. The first category contains recommendations regarding continued development of Paramaterial. The second category contains recommendations that relate to the case studies.

Recommendations for further development of Paramaterial are as follows:

1. *Functionalities related to specific processing requirements of test types not yet covered should be implemented.* The suite of data processing functions provided by Paramaterial should be expanded to include additional test types. Case studies with Jupyter Notebooks should also be provided as tutorials for processing the new test types.
2. *A GUI should be developed for Paramaterial.* The development of a GUI would make the functionalities of Paramaterial available to users with limited coding proficiency.
3. *The use of artificial intelligence for quality control should be explored.* The possibility of using machine learning techniques to automate quality control procedures should be investigated.
4. *Additional material models should be included as in-built Paramaterial models or as functions imported from existing libraries.* Additional material models should be added to the suite of models currently provided by Paramaterial. In particular, models that describe other phenomena, such as softening, would improve the utility of the software.

Recommendations relating to the case studies are as follows:

6. *A Gleeble data post-processing pipeline should be developed using Paramaterial.* The tests in the example datasets used in case studies CS3 and CS4 were performed on the Gleeble 3800, a thermo-mechanical testing machine at CME. This machine will continue to be used to produce mechanical test data. It would therefore be beneficial to develop a machine-specific processing pipeline for data from the Gleeble by leveraging the functionality of Paramaterial. The quality control functionalities could be used during testing to catch anomalies and prevent the production of flawed datasets.
7. *Further analysis should be done using the AA3104 example datasets.* The AA3104 dataset is of particular interest to researchers in the ARG at CME, as this material is manufactured by Hulamin. It would be useful to compare results for the different material states in the datasets: as-cast, homogenised, and transfer-bar; as these states correspond to different industrial processing stages for the alloy. However, it is important to note that the transfer bar data was generated using plain-strain compression tests, while the data for the other material states is from uniaxial compression tests.

7.4 Reflecting on Objectives

All objectives of the project have been satisfactorily met. Arguments substantiating this claim are presented in this section. For convenience, the objectives are re-stated below prior to being addressed.

Obj. 1. *Develop an extensible software toolkit for processing mechanical test data.*

The Paramaterial Python package was developed in this project to satisfy **Obj. 1**. The source code has been open-sourced and is available at https://github.com/dan-slater/paramaterial. The dataset processing example Jupyter Notebooks given in the appendices demonstrate that the software effectively addresses the challenges associated with the processing of mechanical test data.

Obj. 2. *Demonstrate the toolkit functionalities by processing existing mechanical testing datasets.*

The demonstrations in the Jupyter Notebooks also contribute to the fulfilment of **Obj. 2**. Further evidence for the completion of this objective is provided by the usage example. Moreover, the density and richness of information extracted from the example datasets and presented in the case studies is testament to the efficacy of Paramaterial.

Obj. 3. *Document and deploy the toolkit so that other researchers can utilise existing functionalities, as well as modify or extend the software capabilities.*

Comprehensive documentation for Paramaterial has been created and is available at dan-slater.github.io/paramaterial/. The toolkit was also deployed on PyPI, making it easily installable for new users. A final argument for the completion of **Obj. 3** is that the package's modular design, allowing users to combine stand-alone functions in versatile ways, promotes extensibility.

7.5 Closing Remarks

Paramaterial is intended to be a widely-used library for processing mechanical test data, supported by a community of scientists continually contributing to and improving the package. It is hoped that the emphasis that has been placed on repeatability and traceability during the early stages of development will foster a culture of producing high-quality data.

Presently, high-quality open-source mechanical testing datasets are scarce. The software presented in this report has been developed and published with the ideal of improving the material science and mechanical engineering fields in this area.

This dissertation marks the beginning of Paramaterial, but the future of the project depends on the collective efforts of those who join the cause.

References

[1] K. Komvopoulos, *Mechanical Testing of Engineering Materials*. Cognella, 2017.

[2] Y. Brechet, “Physically Based Models for Industrial Materials: What For?,” *Continuum Scale Simulation of Engineering Materials: Fundamentals–Microstructures–Process Applications*, pp. 231–248, 2004.

[3] A. Bhargava and C. Sharma, “Mechanical Properties,” *Mechanical Behaviour and Testing of Materials*, 2011.

[4] S. M. Arnold, F. A. Holland, B. A. Bednarcyk, and E. J. Pineda, “Combining Material and Model Pedigree is Foundational to Making ICME a Reality,” *Integrating Materials and Manufacturing Innovation*, vol. 4, no. 1, pp. 37–62, 2015.

[5] K. Howard and M. Dana, *ASM Handbook Volume 8: Mechanical Testing and Evaluation*, 2000.

[6] W. F. Hosford and R. M. Caddell, “Stress and Strain,” *Metal Forming: Mechanics and Metallurgy*, 2011.

[7] H. Czichos, T. Saito, and L. Smith, *Springer Handbook of Materials Measurement Methods*, vol. 978. Springer, 2006.

[8] S. M. Arnold, F. Holland, T. P. Gabb, M. Nathal, and T. T. Wong, “The Coming ICME Data Tsunami and What Can Be Done,” in *54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, p. 1850, 2013.

[9] S. Arnold, “Paradigm Shift in Data Content and Informatics Infrastructure Required for Generalized Constitutive Modeling of Materials Behavior,” *MRS bulletin*, vol. 31, no. 12, pp. 1013–1021, 2006.

[10] W. Y. Wang, J. Li, W. Liu, and Z.-K. Liu, “Integrated Computational Materials Engineering for Advanced Materials: A Brief Review,” *Computational Materials Science*, vol. 158, pp. 42–48, 2019.

[11] L. Aagesen, J. Adams, J. Allison, W. Andrews, V. Araullo-Peters, T. Berman, Z. Chen, S. Daly, S. Das, S. DeWitt, *et al.*, “Prisms: An Integrated, Open-Source Framework for Accelerating Predictive Structural Materials Science,” *JOM*, vol. 70, no. 10, pp. 2298–2314, 2018.

[12] N. E. Dowling, “Structure and Deformation in Materials,” *Mechanical Behaviour of Materials*, pp. 40–63, 2013.

[13] H. Li and M. Fu, “Introduction to Deformation-Based Manufacturing,” *Deformation-Based Processing of Materials*, pp. 1–28, 2019.

[14] J. Lubliner, “The Physics of Plasticity,” *Plasticity Theory*, pp. 69–101, 1990.

[15] K. K. Sankaran and R. S. Mishra, *Metallurgy and Design of Alloys With Hierarchical Microstructures*. Elsevier, 2017.

[16] M. E. Gurtin, E. Fried, and L. Anand, *The Mechanics and Thermodynamics of Continua*. Cambridge University Press, 2010.

[17] A. Phillion, S. Thompson, S. Cockcroft, and M. Wells, “Tensile Properties of As-Cast Aluminum Alloys AA3104, AA6111 and CA31218 at Above Solidus Temperatures,” *Materials Science and Engineering: A*, vol. 497, no. 1-2, pp. 388–394, 2008.

[18] J. Simo, *Computational Inelasticity*. New York: Springer, 1998.

[19] P. Haupt, *Continuum Mechanics and Theory of Materials*. Advanced texts in physics, Berlin: Springer, 2000.

[20] E. A. de Souza Neto, *Computational Methods for Plasticity: Theory and Applications*. Chichester, West Sussex, UK: Wiley, 2008.

[21] W. Ramberg and W. R. Osgood, “Description of Stress-Strain Curves by Three Parameters,” 1943.

[22] E. Voce, “The Relationship Between Stress and Strain for Homogeneous Deformation,” *Journal of the Institute of Metals*, vol. 74, pp. 537–562, 1948.

[23] A. Alankar and M. A. Wells, “Constitutive Behavior of As-Cast Aluminum Alloys AA3104, AA5182 and AA6111 at Below Solidus Temperatures,” *Materials Science and Engineering: A*, vol. 527, no. 29-30, pp. 7812–7820, 2010.

[24] G. R. Johnson, “A Constitutive Model and Data for Materials Subjected to Large Strains, High Strain Rates, and High Temperatures,” *Proc. 7th Inf. Sympo. Ballistics*, pp. 541–547, 1983.

[25] F. J. Zerilli and R. W. Armstrong, “Dislocation-Mechanics-Based Constitutive Relations for Material Dynamics Calculations,” *Journal of Applied Physics*, vol. 61, no. 5, pp. 1816–1825, 1987.

[26] M. Dundu, “Evolution of Stress-Strain Models of Stainless Steel in Structural Engineering Applications,” *Construction and Building Materials*, vol. 165, pp. 413–423, 2018.

[27] T. Mirzaie, H. Mirzadeh, and J.-M. Cabrera, “A Simple Zerilli–Armstrong Constitutive Equation for Modeling and Prediction of Hot Deformation Flow Stress of Steels,” *Mechanics of Materials*, vol. 94, pp. 38–45, 2016.

[28] C. Zener and J. H. Hollomon, “Effect of Strain Rate Upon Plastic Flow of Steel,” *Journal of Applied Physics*, vol. 15, no. 1, pp. 22–32, 1944.

[29] C. M. Sellars and W. McTegart, “On the Mechanism of Hot Deformation,” *Acta Metallurgica*, vol. 14, no. 9, pp. 1136–1138, 1966.

[30] J. Jonas, C. Sellars, and W. M. Tegart, “Strength and Structure Under Hot-Working Conditions,” *Metallurgical Reviews*, vol. 14, no. 1, pp. 1–24, 1969.

[31] H. Shi, A. McLaren, C. Sellars, R. Shahani, and R. Bolingbroke, “Constitutive Equations for High Temperature Flow Stress of Aluminium Alloys,” *Materials Science and Technology*, vol. 13, no. 3, pp. 210–216, 1997.

[32] S. Davenport, N. Silk, C. Sparks, and C. M. Sellars, “Development Of Constitutive Equations For Modelling Of Hot Rolling,” *Materials Science and Technology*, vol. 16, no. 5, pp. 539–546, 2000.

[33] M. Rout, S. K. Pal, and S. B. Singh, “Finite element modeling of Hot Rolling: Steady-And Unsteady-State Analyses,” in *Computational Methods and Production Engineering*, pp. 83–124, Elsevier, 2017.

[34] G. Guo, *Aluminum Microstructure Evolution and Effects on Mechanical Properties in Quenching and Aging Process*. PhD thesis, Worcester, 2017.

[35] M. F. Novella, A. Ghiotti, and S. Bruschi, “Effect of the Zener-Hollomon Parameter on the Formability of AA6082-T6 Formed at Elevated Temperature,” in *AIP Conference Proceedings*, AIP Publishing LLC, 2016.

[36] T. Li, B. Zhao, X. Lu, H. Xu, and D. Zou, “A Comparative Study on Johnson Cook, Modified Zerilli–Armstrong, and Arrhenius-Type Constitutive Models to Predict Compression Flow Behavior of SnSbCu Alloy,” *Materials*, vol. 12, no. 10, p. 1726, 2019.

[37] X. Wang, K. Chandrashekhara, S. A. Rummel, S. Lekakh, and D. C. Van Aken, "Modeling of Mass Flow Behavior of Hot Rolled Low Alloy Steel Based on Combined Johnson-Cook and Zerilli-Armstrong Model," *Journal of Materials Science*, vol. 52, no. 5, pp. 2800–2815, 2017.

[38] Industrial Physics, "Electromechanical vs Servo-Hydraulic UTMs." https://industrialphysics.com/knowledgebase/articles/electromechanical-vs-servo-hydraulic-utms/. [Online; Accessed 15-September-2022].

[39] Instron, *Universal Testing Machine Compliance*, 2010.

[40] Y. Bergström, "The Plastic Deformation Process Of Metals–50 Years Development Of A Dislocation Based Theory," 2015.

[41] M. Khadyko, S. Dumoulin, T. Børvik, and O. S. Hopperstad, "Simulation of Large-Strain Behaviour of Aluminium Alloy Under Tensile Loading Using Anisotropic Plasticity Models," *Computers & Structures*, vol. 157, pp. 60–75, 2015.

[42] X. Hu, M. Jain, D. Wilkinson, and R. Mishra, "Microstructure-Based Finite Element Analysis of Strain Localization Behavior in AA5754 Aluminum Sheet," *Acta Materialia*, vol. 56, no. 13, pp. 3187–3201, 2008.

[43] B. Roebuck, J. D. Lord, M. Brooks, M. S. Loveday, C. M. Sellars, and R. W. Evans, "Measuring Flow Stress in Hot Axisymmetric Compression Tests," 2002.

[44] M. Pietrzyk, J. Lenard, and G. Dalton, "A Study of the Plane Strain Compression Test," *CIRP annals*, vol. 42, no. 1, pp. 331–334, 1993.

[45] N. Silk and M. Van Der Winden, "Interpretation of Hot Plane Strain Compression Testing of Aluminium Specimens," *Materials Science and Technology*, vol. 15, no. 3, pp. 295–300, 1999.

[46] M. Loveday, G. Mahon, B. Roebuck, A. Lacey, E. Palmiere, C. Sellars, and M. Van der Winden, "Measurement of Flow Stress in Hot Plane Strain Compression Tests," *Materials at High Temperatures*, vol. 23, no. 2, pp. 85–118, 2006.

[47] D. S. Inc., *Gleeble Application Notes*, 2017.

[48] R. Hand, S. Foster, and C. Sellars, "Temperature Changes During Hot Plane Strain Compression Testing," *Materials Science and Technology*, vol. 16, no. 4, pp. 442–450, 2000.

[49] M. R. van der Winden, *Laboratory Simulation and Modelling of the Break-Down Rolling of AA3104*. PhD thesis, University of Sheffield, 1999.

[50] M. Loveday, G. Mahon, B. Roebuck, C. Sellars, and M. van der Winden, "Measuring Flow Stress in Plane Strain Compression Tests," 2002.

[51] B. Roebuck, J. Lord, M. Brooks, M. Loveday, C. Sellars, and R. Evans, "Measurement of Flow Stress in Hot Axisymmetric Compression Tests," *Materials at High Temperatures*, vol. 23, no. 2, pp. 59–83, 2006.

[52] S. P. Mates, R. Rhorer, E. Whitenton, T. Burns, and D. Basak, "A Pulse-Heated Kolsky Bar Technique for Measuring the Flow Stress Of Metals at High Loading and Heating Rates," *Experimental Mechanics*, vol. 48, no. 6, pp. 799–807, 2008.

[53] B. Roebuck, M. Brooks, and M. Gee, "Load Cell Ringing in High Rate Compression Tests," *Applied Mechanics and Materials*, 2004.

[54] ASTMInternational, "Standard Test Methods for Tension Testing of Metallic Materials," *ASTM*, 2001.

[55] B. Aakash, J. Connors, and M. D. Shields, "Variability in the Thermo-Mechanical Behavior of Structural Aluminum," *Thin-Walled Structures*, vol. 144, p. 106122, 2019.

[56] D. A. Morrow, T. Haut Donahue, G. M. Odegard, and K. R. Kaufman, "A Method for assessing the fit of a Constitutive Material Model to Experimental Stress–Strain Data," *Computer Methods In Biomechanics And Biomedical Engineering*, vol. 13, no. 2, pp. 247–256, 2010.

[57] A. International, "Standard Test Method for Young's Modulus, Tangent Modulus, and Chord Modulus," 2010.

[58] W. Gabauer, "The Determination of Uncertainties in Tensile Testing," *Manual of Codes of Practice for the Determination of Uncertainties in Mechanical Tests on Metallic Materials*, 2000.

[59] L. L. Yaw, "Nonlinear Static 1D Plasticity: Various Forms of Isotropic Hardening," *Walla Walla University*, 2012.

[60] B. Aakash, J. Connors, and M. D. Shields, "Stress-Strain Data for Aluminum 6061-T651 from 9 Lots at 6 Temperatures Under Uniaxial and Plane Strain Tension," *Data in brief*, vol. 25, p. 104085, 2019.

[61] S. G. Shanle Baron, "Hulamin Progress Report," tech. rep., University of Cape Town, 2018.

[62] C. Hyde, "Critical Analysis of Simulated Thermomechanical Processing of Aluminium Can Body Stock," Master's thesis, University of Cape Town, 2015.

[63] R. E. Smallman and A. Ngan, *Physical Metallurgy and Advanced Materials*. Elsevier, 2007.

[64] C. Sellars, J. Sah, J. Beynon, and S. Foster, "Report on Research Work Supported by Science Research Council Grant No RG/1481," 1976.

[65] M. Mirza and C. Sellars, "Modelling the Hot Plane Strain Compression Test Part 2–Effect of Friction and Specimen Geometry on Spread," *Materials Science and Technology*, vol. 17, no. 9, pp. 1142–1148, 2001.

[66] H. Shi, A. McLaren, C. Sellars, R. Shahani, and R. Bolingbroke, "Hot Plane Strain Compression Testing of Aluminum Alloys," *Journal of Testing and Evaluation*, vol. 25, no. 1, pp. 61–73, 1997.

A Additional Mechanical Testing Derivations

A.1 Ideal Plane-Strain Conditions

A yield criterion can be used to compare the three-dimensional stress state of a material to an equivalent scalar stress, $\bar{\sigma}$. The Tresca or von Mises criteria are commonly used for this purpose. The Tresca criterion postulates that yielding depends on the largest shear stress in a body [6], and can be expressed in terms of principal stresses and the yield strength in uniaxial loading or shear,

$$\bar{\sigma} = \max(|\sigma_1 - \sigma_2|, |\sigma_2 - \sigma_3|, |\sigma_1 - \sigma_3|)\,. \tag{A.1}$$

The von Mises criterion is based on the theory of distortional strain energy [7], and is expressed as

$$\bar{\sigma} = \frac{\sqrt{2}}{2}\sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2} \tag{A.2}$$

For isotropic materials[2] under loading with a constant ratio of $d\varepsilon_1 : d\varepsilon_2 : d\varepsilon_3$[3], an equivalent strain, $\bar{\varepsilon}$, can also be defined [6, 63]. Using the Tresca criterion, the equivalent strain is

$$\bar{\varepsilon} = max(|\varepsilon_1|, |\varepsilon_2|, |\varepsilon_3|)\,, \tag{A.3}$$

and using the von Mises criterion,

$$\bar{\varepsilon} = \frac{\sqrt{2}}{3}\sqrt{(\varepsilon_1 - \varepsilon_2)^2 + (\varepsilon_2 - \varepsilon_3)^2 + (\varepsilon_3 - \varepsilon_1)^2}\,. \tag{A.4}$$

For a material subjected to a multi-axial load, the equivalent stress can be compared to the yield strength, σ_Y. Then, yielding is expected if

$$\bar{\sigma} \geq \sigma_Y\,. \tag{A.5}$$

In a material subjected to a uniaxial load, for example in the uniaxial tensile test, only σ_1 is non-zero, so that both the Tresca and von Mises criteria reduce to

$$\bar{\sigma} = \sigma_1\,. \tag{A.6}$$

[2] An isotropic material is a material that has the same properties in all directions.
[3] ε_1, ε_2 and ε_3 are principal strains.

Hence, the equivalent scalar stress, $\bar{\sigma}$, is also called the equivalent tensile stress.

If a stress-strain curve is plotted from plane-strain test measurements, see Figure 2.4, the yield strength will appear to be higher than that measured using a uniaxial tension test. Poisson's ratio,

$$\nu = -\frac{\varepsilon_{transverse}}{\varepsilon_{axial}} = -\frac{\varepsilon_y}{\varepsilon_x} = -\frac{\varepsilon_z}{\varepsilon_x} , \tag{A.7}$$

can be used to relate the yield stress during multi-axial loading to the yield stress from a uniaxial tensile test [5]. A force, F, applied in the x-direction will generate an axial stress

$$\sigma_x = \frac{F}{A} , \tag{A.8}$$

where A is the cross-sectional area of the gauge section. The corresponding axial strain, σ_x, will be

$$\varepsilon_x = \frac{\sigma_x}{E} . \tag{A.9}$$

Additionally, there will be transverse strains, ε_y and ε_z,

$$\varepsilon_y = \varepsilon_z = -\nu\varepsilon_x . \tag{A.10}$$

The three-dimensional expressions relating elastic stresses and strains are

$$\varepsilon_x = \frac{\sigma_x - \nu\sigma_y - \nu\sigma_z}{E} , \tag{A.11}$$

$$\varepsilon_y = \frac{\sigma_y - \nu\sigma_z - \nu\sigma_x}{E} , \tag{A.12}$$

$$\varepsilon_z = \frac{\sigma_z - \nu\sigma_x - \nu\sigma_y}{E} . \tag{A.13}$$

When $\varepsilon_y = 0$ and $\sigma_z = 0$, equation (A.12) becomes

$$\varepsilon_y = 0 = \frac{\sigma_y - \nu\sigma_x}{E} , \tag{A.14}$$

so that stresses in the x and y directions can be related by

$$\sigma_y = \nu\sigma_x . \tag{A.15}$$

Substituting $\sigma_1 = \sigma_x$, $\sigma_3 = \sigma_z = 0$, and $\sigma_2 = \sigma_y = \nu\sigma_x$ into equation (A.2) leads to the yield criterion for plane-strain tension,

$$(\sigma_x - \nu\sigma_x)^2 + (\nu\sigma_x)^2 + (-\sigma_x)^2 = 2\bar{\sigma}^2 , \tag{A.16}$$

which simplifies to

$$\bar{\sigma} = \sigma_x\sqrt{1 - \nu + \nu^2} . \tag{A.17}$$

For an incompressible material during plastic flow, $\nu = \frac{1}{2}$, so that the plane-strain axial stress can be related to the equivalent tensile stress by

$$\bar{\sigma} = \frac{\sqrt{3}}{2}\sigma_x . \tag{A.18}$$

During plastic flow, conservation of volume can be used to determine an expression for the equivalent strain. Consider a cube in the specimen's centre with volume $V = xyz$. In plane-strain, if $dy = 0$,

$$(x + \mathrm{d}x)y(z + \mathrm{d}z) = xyz\,, \tag{A.19}$$

$$\frac{(x + \mathrm{d}x)}{x} = \frac{z}{(z + \mathrm{d}z)}\,, \tag{A.20}$$

$$\ln\left(\frac{x + \mathrm{d}x}{x}\right) = -\ln\left(\frac{z + \mathrm{d}z}{z}\right)\,, \tag{A.21}$$

$$\varepsilon_x = -\varepsilon_z\,. \tag{A.22}$$

Substituting $\varepsilon_y = \varepsilon_2 = 0$ and equation (A.22), where $\varepsilon_x = \varepsilon_1$ and $\varepsilon_z = \varepsilon_3$, into equation (A.4), results in

$$\bar{\varepsilon} = \frac{2}{\sqrt{3}}\varepsilon_x\,. \tag{A.23}$$

A.2 Friction Correction for Uniaxial Compression Tests

Friction correction is necessary during post-processing of uniaxial compression test data [43, 51]. After calculating the pressure using the current diameter, D,

$$P = \frac{4F}{\pi D^2}\,, \tag{A.24}$$

the friction-corrected true stress can be determined using [43]

$$\sigma = P\left[\frac{\mu D}{L}\right]^2\left[\exp\left(\frac{\mu D}{L}\right) - \frac{\mu D}{L} - 1\right]^{-1}\,, \tag{A.25}$$

where μ is the coefficient of friction and L is the current height For low values of μ, this can be simplified to [43]

$$\sigma = P\left[\frac{1 + \mu D}{3L}\right]\,. \tag{A.26}$$

A.3 Corrections for Plane-Strain Compression Tests

A.3.1 Breadth-Spread Correction

Plane-strain compression tests tend to exhibit bulging, so that the breadth[1] of the specimen cannot be considered constant throughout the test [45]. An empirically determined breadth spread coefficient [45, 64, 65], can be calculated as

$$C = \frac{(b_f/b_0) - 1}{1 - (h_f/h_0)^n} \tag{A.27}$$

where b_0, b_f, h_0, and h_f are the initial breadth, final breadth, initial height, and final height, respectively, and n is a constant that should be determined for different lubricant and material combinations. The instantaneous breadth, b, is then given by

$$b = b_0 \left[1 + C - C \left(\frac{h}{h_0} \right)^n \right] . \tag{A.28}$$

When equation (A.27) was first reported in [64], the authors suggested a value of $n = 0.5$. Subsequent studies have used $n = 0.2$ for oil-based graphite lubricants and aluminium specimens [66], and $n = 0.18$ if water-based graphite lubricants are used instead [45]. However, if a consistent value for n is used in equations (A.27) and (A.28), differences in its magnitude have minimal effect [46].

The strain in the y-direction[2] can then be calculated as

$$\varepsilon_y = \ln \left(\frac{b}{b_0} \right) , \tag{A.29}$$

and strain in the x-direction is still calculated using

$$\varepsilon_x = \ln \left(\frac{L_0 - \Delta L}{L_0} \right) \tag{A.30}$$

Furthermore, Poisson's ratio is $\nu = 0.5$ for metals during plastic flow [5]. Thus,

$$\nu = \frac{1}{2} = -\frac{\varepsilon_x}{\varepsilon_z} = -\frac{\varepsilon_y}{\varepsilon_z} , \tag{A.31}$$

so that

$$\varepsilon_z = -\varepsilon_y - \varepsilon_x . \tag{A.32}$$

[1] Denoted by b in Figure 2.6.
[2] See Figure 2.6.

Substituting (A.32) into (A.2), leads to

$$\bar{\varepsilon} = \frac{2}{\sqrt{3}}\sqrt{\varepsilon_x^2 + \varepsilon_x\varepsilon_y + \varepsilon_y^2}\,. \tag{A.33}$$

A.3.2 Friction Correction

To determine the equivalent flow stress using plane strain compression tests, measurements must be corrected for friction and breadth spread [45].

The average pressure at the interface is given by

$$\bar{p} = \frac{F}{wb}\,. \tag{A.34}$$

Depending on the extent of deformation, friction at the interface may be sliding, sticking or a combination of the two. Solving a differential equation for the pressure distribution in the z-direction[1] produces an equation for the distance, z_0, from the centre to the position where sliding changes to sticking [45],

$$z_0 = \left(\frac{h}{2\mu}\right)\ln\left(\frac{1}{2\mu}\right)\,. \tag{A.35}$$

For full sliding friction, $2z_0 > w$, and the yield stress in pure shear, k, can be related to the average pressure by [45]

$$\frac{\bar{p}}{2k} = \frac{1}{bw}\left[\frac{2h^2}{\mu^2} + \frac{(b-w)h}{\mu}\right]\left[\exp\left(\frac{\mu w}{h}\right) - 1\right] - \frac{2h}{\mu b}\,. \tag{A.36}$$

If $w > 2z_0 > 0$, then there is partial sticking-sliding friction, so that [45]

$$\begin{aligned}\frac{\bar{p}}{2k} = {} & \frac{\left(\frac{w}{2} - z_0\right)}{\mu w} + \frac{\left(\frac{w}{2} - z_0\right)^2}{hw} + \frac{h}{\mu w}\left(\frac{1}{2\mu} - 1\right) + \frac{1}{hb}\left(z_0^2 - \frac{4z_0^3}{3w} - \frac{w^2}{12}\right) \\ & + \frac{1}{\mu b}\left(\frac{2z_0^2}{w} - z_0 - \frac{2hz_0}{\mu w} + \frac{h}{2\mu} - h + \frac{h^2}{w\mu^2} - \frac{2h^2}{\mu w}\right)\,.\end{aligned} \tag{A.37}$$

If $0 > 2z_0$, full sticking friction prevails, and [45]

$$\frac{\bar{p}}{2k} = 1 + \frac{w}{4h} - \frac{w^2}{12hb}\,. \tag{A.38}$$

The equivalent flow stress can then be calculated as

$$\bar{\sigma} = \frac{2k}{f}\,, \tag{A.39}$$

[1] see Figure 2.6.

where f is a factor used to account for breadth spread [46].

A.3.3 Isothermal Correction

The Zener-Holloman model can be used to perform isothermal correction. To do so, use equation (2.22) to determine values for the material parameter, B, at several different nominal strains, ε_{nom}. Then, fit a power law to obtain an expression for the material parameter as a function of nominal strain [50],

$$B(\varepsilon_{nom}) = a\varepsilon_{nom}^{b} \tag{A.40}$$

where the constants a and b are fitted. The corrected stress can then be calculated as a function of nominal strain using [50]

$$\sigma_{iso}(\varepsilon_{nom}) = \sigma^{*} + \frac{Q}{B(\varepsilon_{nom})R}\left(\frac{1}{T_{iso}} - \frac{1}{T_{inst}}\right) \tag{A.41}$$

where σ^{*} is the uncorrected stress, σ_{iso} is the corrected stress, T_{iso} is the desired isothermal temperature, T_{inst} is the temperature recorded with a thermocouple, Q is the activation energy, and R is the universal gas constant.

B Additional Implementation Details

B.1 Architecture of Paramaterial

The architecture of Paramaterial was designed as a library of functions and classes, emulating the style of widely used Python data science libraries, such as NumPy, Pandas and Matplotlib. This approach was chosen to maximise the versatility of Paramaterial, allowing users to easily customize and extend its functionality as needed.

The distinct functions and classes provided by Paramaterial can be employed to execute individual tasks required when processing and analysing mechanical test datasets. Developed as independent, stand-alone tools, these functions and classes can then be combined in versatile manners to accomplish further unanticipated tasks.

A drawback of offering these stand-alone tools is that each function or class must be documented. Furthermore, the documentation requires ongoing maintenance as the code library evolves. Nevertheless, it was determined that the versatility of this architectural style resulted in a high likelihood of widespread adoption, making the required maintenance justifiable.

Two primary alternative architecture styles were assessed: the configuration file approach and the graphical user interface (GUI) approach. Each possesses its own merits and drawbacks in comparison to the chosen style.

The configuration file approach allows for an automated process without requiring Python expertise. However, its drawbacks include the potential for limited flexibility in accommodating specific user requirements and the need for users to learn the configuration file syntax, which might be less intuitive than a Python-based interface.

The GUI approach offers a user-friendly experience, making it easier for to navigate the software and use existing functionalities. However, the drawbacks of a GUI approach include limited customisability and extensibility, difficulty of integration with other tools, and increased time and resources needed for development and maintenance.

Both alternatives demonstrate constraints in terms of versatility and maintainability. The chosen architecture, however, attains a balance between user-friendliness and the ability to address distinct and unanticipated challenges.

B.2 Contents and Structure of the Repository

The Paramaterial project is hosted on GitHub. The repository contains all of the code, documentation, and other files necessary to build and run the package. The repository is organized into several top-level directories, each of which contains subdirectories and files relevant to the project, as illustrated in Figure B.1. The purpose of each file or folder is outlined as follows:

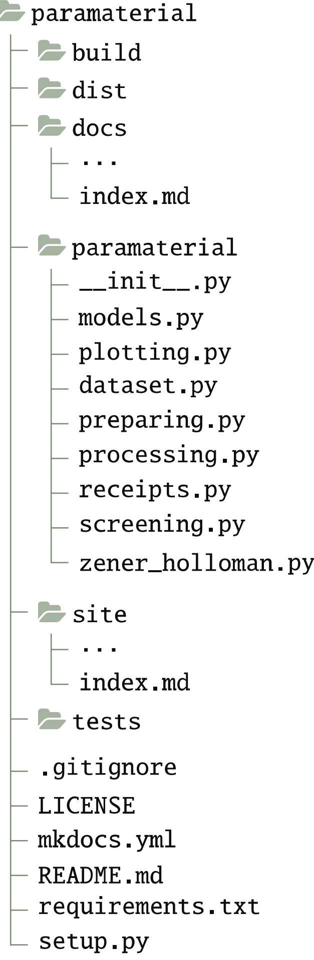


Figure B.1: *File tree showing structure and contents of the Paramaterial project repository.*

- The `build` folder contains the files necessary for creating a build of the package, allowing for easy distribution and installation.
- The `dist` folder holds the built distribution files, ready for distribution and installation.
- The `docs` folder contains documentation for the package, including an `index.md` file with information on how to use the package. This folder is important for ensuring users can understand and properly utilize the package.
- The `paramaterial` folder is the heart of the package and contains all the source code for the package.

- The `init.py` file is a special Python file that allows the package to be treated as a package, rather than a collection of individual modules.
- The `models.py` file contains the classes and functions that define the package's models.
- The `plotting.py` module contains functions for creating plots, `dataset.py` contains functions for handling datasets, `preparing.py` contains functions for preparing data for processing, `processing.py` contains functions for processing the data, `receipts.py` contains functions for creating receipts, and `screening.py` contains functions for screening the data.
- The `zener_holloman.py` module contains functions for modeling and fitting the data using the Zener-Hollomon model. This will be merged with the `model.py` module in a future release.
- The `site` folder contains the source files for the package's documentation website, including an `index.md` file, providing users with clear, concise, and easily accessible documentation.
- The `tests` folder contains the test suite for the package, ensuring that the package functions correctly and reliably.
- The `.gitignore` file lists files and directories that should be ignored by Git, the version control system used for the package's development.
- The `LICENSE` file contains the package's license, outlining the terms of use for the package.
- The `mkdocs.yml` file contains the configuration for the package's documentation website.
- The `README.md` file contains a brief overview of the package, its purpose, and how to use it.
- The `requirements.txt` file lists the dependencies required to run the package, containing a comprehensive list of the Python packages needed to operate Paramaterial.
- The `setup.py` file contains the package's metadata and is used to build and distribute the package, including the installation script for the "Paramaterial" package.

The repository structure is designed to be clear and well-organized, making it easy for contributors to find and modify the relevant files. The repository structure reflects the best practices and conventions of the Python community, as outlined in[1] PEP 423.

The Package paramaterial has been outlined in this section.

[1] A Python Enhancement Proposal (PEP) is a design document that proposes new features or improvements to the Python programming language or its ecosystem. Once a PEP is accepted, it becomes an official part of the Python documentation, and its proposals are expected to be implemented by Python developers.

B.3 Important Classes

The `DataSet` and `DataItem` classes were designed to make processing large sets of time-series data as easy as processing a single test. The classes facilitate bulk processing. The `DataSet` and `DataItem` setup was designed so that the user could work on code to manipulate a single `DataItem`, and then easily apply that to the entire `DataSet`.

Each `DataItem` contains the experimental measurements for a test in a `Pandas.DataFrame`, the information for that test in a `Pandas.Series`, and a string containing the test ID. The `DataItem` class was designed as a container for data, and the functionality for handling `DataItem` instances was built into the `DataSet` class, see Figure B.2.

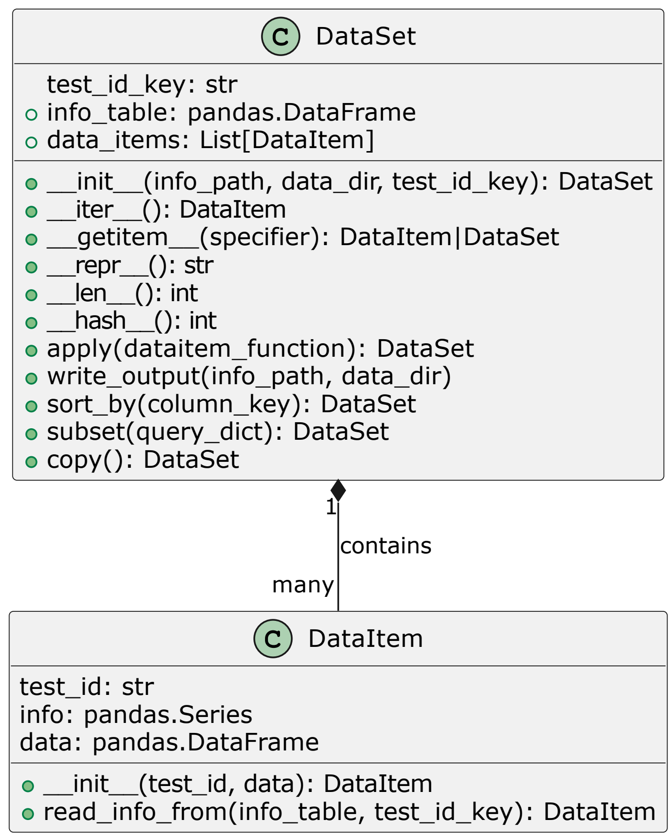


Figure B.2: *UML class diagram for the data handling classes.*

The `Styler` class is used to handle styling for the Paramaterial plotting functions, see Figure B.3. The various attributes are used to store different formatting information for plotting. The formatters can be adjusted to match the contents of a given `DataSet` using the `Styler.style_to()` function. The appropriate curve formatters for a given `DataItem` can then be obtained using the `Styler.curve_formatters()` function. The `Styler` object also handles making the legend for the plots. The `Styler.legend_handles()` function can be used to a list of the automatically generated legened entries, should the user wish to customise the legend.

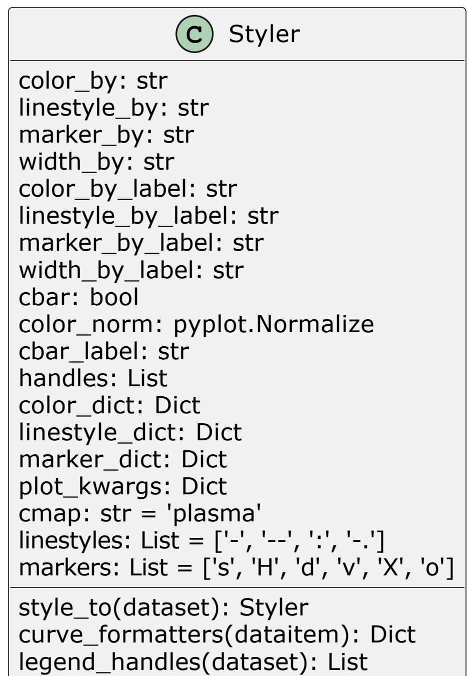


Figure B.3: *UML class diagram for the Styler class.*

The `ModelSet` class is used for fitting a model to the time-series data in a dataset. Once fitted, an artificial dataset class can be generated. Figure B.4. The `ModelSet` class is used by first initialise the class and passing in a model function.

```
ModelSet
  model_func: Callable[[numpy.ndarray, Tuple[float]], numpy.ndarray]
  x_key: str
  y_key: str
  param_keys: List[str]
  var_keys: List[str]|None
○ fitting_results: pandas.DataFrame
● __init__(model_func, x_key, y_key, param_keys, var_keys): ModelSet
● _objective_function(param_vals, dataitem)
● _fit_item(dataitem)
● fit_to(dataset, resolution, optimiser, kwargs): ModelSet
● predict(x_min, x_max, resolution): DataSet
● from_info_table(info_table, model_func, param_keys, var_keys): ModelSet
```

Figure B.4: *UML class diagram for the model fitting class.*

The `param_names` describe the parameters that will be optimised, the `var_keys` describe `info_table` headers that are variables in the model. When fitting a model to a `DataItem`, these `var_keys` will be used to look up the values of the variables for that specific `DataItem`.

Once the `ModelSet` is initialised, a `scipy.optimise` function can be used to run the optimisation. The optimisation function will be passed an objective function. The currently implemented objective

function uses the RMSE error described in equation (2.26).

After fitting, the `ModelSet.predict()` function can be used to generate a new `DataSet`. The predicted `DataSet` will contain artificial data generated using the model, all of the metadata that was previously stored in the `DataSet` to which the `ModelSet` was fit, and any additional values for the optimised model parameters.

The `TestReceipts` class is used for generating a set of test reports, each of which contains information about a given test. See Figure B.5. The function `TestReceipts.parse_placeholders()` can be employed for the purpose of parsing the placeholders contained within the test receipt template. To generate a series of test receipts for a `DataSet` object, the function `TestReceipts.generate_receipts()` is then used. This uses user-specified functions to replace placeholders in the template with test-specific strings, tables or plots.

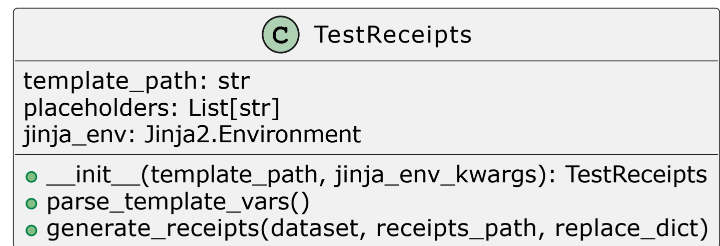


Figure B.5: *UML class diagram for the test receipt generation class.*

B.4 Important Functions

This section contains a list of the important functions that are currently implemented in the Paramaterial package. The descriptions are brief, and the user is referred to the code reference section of the documentation for more information.

- `experimental_matrix()`: This function is used to generate a table showing the number of tests that have been performed for each combination of experimental variables. The function can generate a heatmap, which provides a useful visualisation of the distribution of tests.
- `DataSet.__init__()`: To read in data when performing an analysis using Paramaterial, the `DataSet` class is used. The `DataSet` class is initialised by passing in a path to a folder containing the data files. The data files are read in and stored in the `DataSet` object.
- `DataSet.apply()`: The `DataSet.apply()` function is used to apply a function to each `DataItem` in the `DataSet`. The function can be used to perform calculations on the data, or to generate new data.
- `DataSet.write_to()`: The `DataSet.write_to()` function is used to write the data in the `DataSet` to a folder. The data is written to a file for each `DataItem` in the `DataSet`, and

the combined metadata is written to a spreadsheet.

- `Styler.style_to()`: The `Styler.style_to()` function is used to generate a `Styler` object, which is used to style the plots generated by the `dataset_plot()` and `dataset_subplots()` functions.

- `dataset_plot()`: The `dataset_plot()` function is used to generate a plot of the data in a `DataSet`. The plot can be customised using the `Styler` object.

- `dataset_subplots()`: The `dataset_subplots()` function is used to generate a set of subplots of the data in a `DataSet`. The subplots can be customised using the `Styler` object.

- `find_upl_and_lpl()`: The `find_upl_and_lpl()` function is used to find the upper and lower proportional limits of a stress-strain curve. The function is used to find the limits of the elastic region of the curve.

- `correct_foot()`: The `correct_foot()` function is used to correct the foot of a stress-strain curve. The foot of the curve is corrected by shifting the curve so that a line through the elastic region cuts the origin.

- `generate_screening_pdf()`: The `generate_screening_pdf()` function is used to generate an interactive screening PDF.

- `read_screening_pdf()`: The `read_screening_pdf()` function is used to read in the user entries in the fields of a screening PDF. The entries are read in and stored in a `DataSet` object.

- `make_representative_data()`: The `make_representative_data()` function is used to generate average curves. The representative data is generated by taking the mean of the data for each combination of experimental variables.

- `ModelSet.fit_to()`: The `ModelSet.fit_to()` function is used to fit a set of models to a `DataSet` object. The models are fitted using the `ModelSet.fit_model()` function.

- `ModelSet.predict()`: The `ModelSet.predict()` function is used to generate a `DataSet` object containing artificial data.

- `conformance_matrix()`: The `conformance_matrix()` function is used to generate a conformance matrix for a model fitted to a dataset.

- `TestReceipts.parse_placeholders()`: The `TestReceipts.parse_placeholders()` function is used to parse the placeholders in the test receipt template.

- `TestReceipts.generate_receipts()`: The `TestReceipts.generate_receipts()` function is used to generate a set of test receipts for a `DataSet` object. The placeholders are replaced with the values of the experimental variables.

B.5 Clickable GUI for Selecting Points

In addition to the functionality for automatically identifying points-of-interest on the stress-strain curve, a graphical user interface (GUI) for manual selection of points was developed. The GUI consists of a single window with two plots and two tables, see Figure B.6.

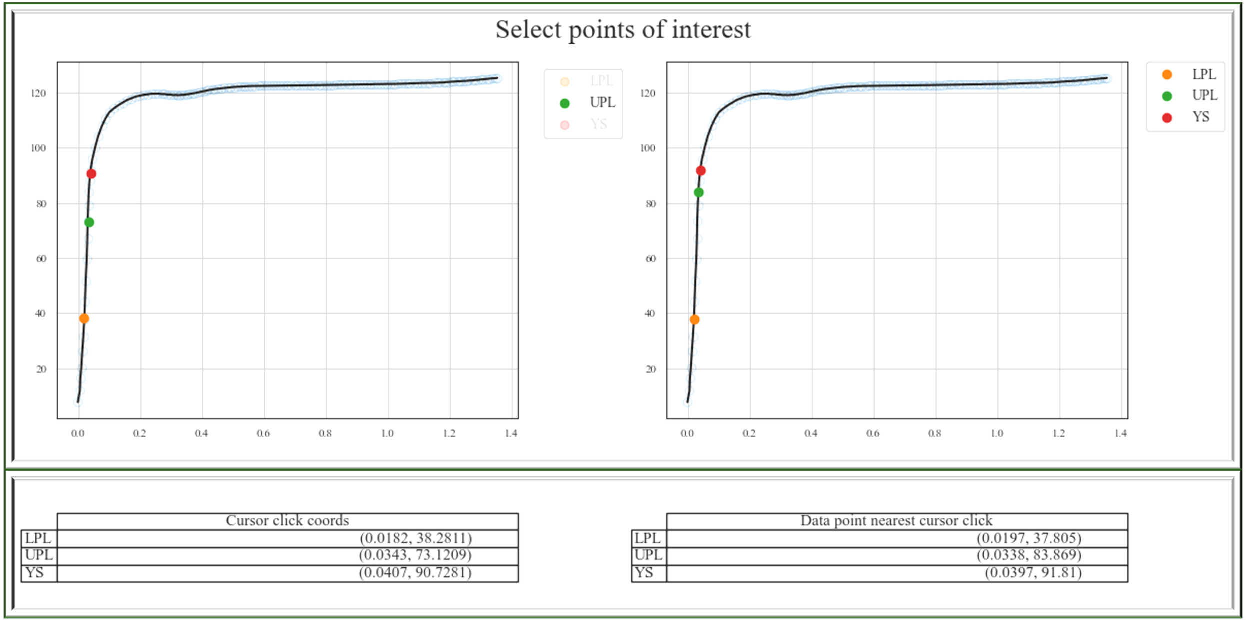


Figure B.6: *The graphical user interface developed for manually selecting points of interest on the stress-strain curve.*

The following steps describe how to use the GUI:

1. Specify a list of names of points to be identified.
2. Pass a `DataSet` object to the GUI function and run it. This will bring up the window.
3. Click on one of the legend entries in the left plot to define which point will be selected and stored.
4. Click on the curve in the left plot, the coordinates of the click will appear next to the name of the point in the bottom-left table.
5. The code will determine which of the actual data points is closest to where the user has clicked. This will be displayed in the plot and table on the right.
6. Once coordinates for all of the points have been found, close the window. This will store the coordinates of the points under their names in the `info` attribute of the current `DataItem`.
7. The window will reappear automatically, displaying the data from the next `DataItem` in the `DataSet`.
8. Repeat steps 3 to 7 until the `DataSet` is exhausted.

B.6 Branding

Part of developing an open-source software package is to make it attractive to potential users and contributors. Moreover, the project's purpose should be readily apparent at first glance. To aid with these goals, a suitable logo was created, see Figure B.7.

Figure B.7: *Paramaterial logo with badges showing current information about the package. This appears at the top of the README in the GitHub repo, and on the PyPI page.*

The Paramaterial logo includes the name of the package and an icon for the package. Both are related to the intended usage. The name is a play on "parameterising material data", which refers to aggregating time-series measurements from mechanical tests into a table of mechanical properties and material parameters. The icon is an artists rendition of a typical set of stress-strain curves, and details of its design are given in Figure B.8.

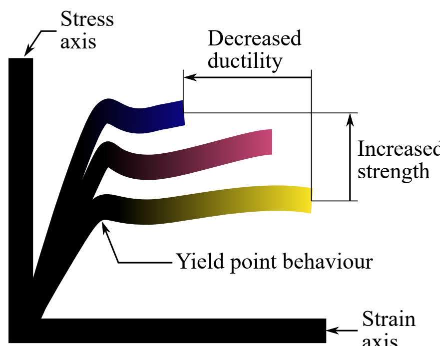


Figure B.8: *Paramaterial icon design. The stress-strain curves show the typical behaviour of decreased ductility with increased strength.*

Informative "badges" are displayed beneath the logo. Badges allow users to easily and immediately assess the compatibility, quality, and stability of a package. Most badges are automatically updated based on information that is collected and processed by third-party services. Specifically, the badges underneath the Paramaterial logo portray the following information:

- The `python: 3` badge indicates that Paramaterial is compatible with Python 3, which is the latest version of the language.
- The `license: MIT` badge shows that Paramaterial is released under the MIT License.
- The `wheel: yes` badge means that Paramaterial can be easily installed and distributed as a

Python wheel[1] package.

- The `dependencies: up to date` badge lets users know that all of the required dependencies for Paramaterial are up to date and compatible with the current version.
- The `pypi package: 0.1.0` badge indicates that the latest stable version of Paramaterial available on PyPI is version 0.1.0.
- The `release: v0.1.1-beta` badge shows that the current version of Paramaterial is a beta release with version number 0.1.1.

When the Paramaterial logo is displayed at the top of its GitHub README file, its PyPI description page, and the home page of its project documentation, the badges provide clickable links to verify the information they represent. In addition to fulfilling a functional purpose, the Paramaterial logo and badges also serve an important aesthetic role in attracting users and contributors to the project. A tone of professionalism and credibility can be established by presenting a clear and visually appealing branding. This helps to build and strengthen the community around the open-source project.

[1] The wheel format is commonly used for distributing Python packages, as it offers pre-built binaries that can be installed without the need for a build step during installation. This feature can save time and minimize installation errors.

B.7 Paramaterial Usage Example Notebook

Importing the paramaterial module

[1]:
```
import pandas as pd
# import paramaterial as a module
import paramaterial as pam

# check the version of paramaterial
print(pam.__version__)
```

```
0.1.0
```

Load the raw data and metadata into a DataSet object

[2]:
```
# load the metadata spreadsheet and data files into a DataSet object
prepared_ds = pam.DataSet('info/01 prepared info.xlsx', 'data/01 prepared data')

# check the formatting of the loaded data and metadata
pam.check_formatting(prepared_ds)
```

```
Checking column headers...
First file headers:
        ['Strain', 'Stress_MPa']
Headers in all files are the same as in the first file, except for None.
Checking for duplicate files...
No duplicate files found in "data/01 prepared data".
```

Sort and Get the metadata as a pandas DataFrame

[3]:
```
# sort the dataitems in the dataset
prepared_ds = prepared_ds.sort_by(['temperature', 'lot'])

# get the metadata as a table
# prepared_ds.info_table
```

Use the subset method to get a subset of the data

[4]:
```
# get a subset of only the tensile tests
prepared_ds = prepared_ds.subset({'test_type': ['T']})
```

Make an experimental matrix showing the distribution of the data

[5]:
```
import matplotlib as mpl
cmap_green = mpl.colors.LinearSegmentedColormap.from_list("", ["white", (0.2124, 0.3495, 0.1692)])

# make a heatmap showing the distribution across lot and temperature
pam.experimental_matrix(info_table=prepared_ds.info_table, index='temperature', vmax=7,
                        columns='lot', as_heatmap=True, cmap=cmap_green);
```

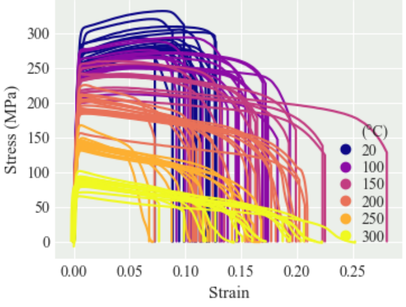
temperature \ lot	A	B	C	D	E	F	G	H	I
20	3	3	3	3	3	1	1	1	1
100	3	3	3	3	3	1	1	1	1
150	0	2	2	2	2	1	1	1	1
200	3	3	3	3	3	1	1	1	1
250	0	2	2	2	2	1	1	1	1
300	3	3	3	3	3	1	1	1	1

Use a Styler object to format the plotting to the entire dataset

[6]:
```
# instantiate a Styler object and format to match the prepared data
styler = pam.Styler(color_by='temperature', color_by_label='(°C)', cmap='plasma')
styler.style_to(prepared_ds)

# keys labels for stress-strain data plotting
labels_dict = dict(x='Strain', y='Stress_MPa', ylabel='Stress (MPa)')
```

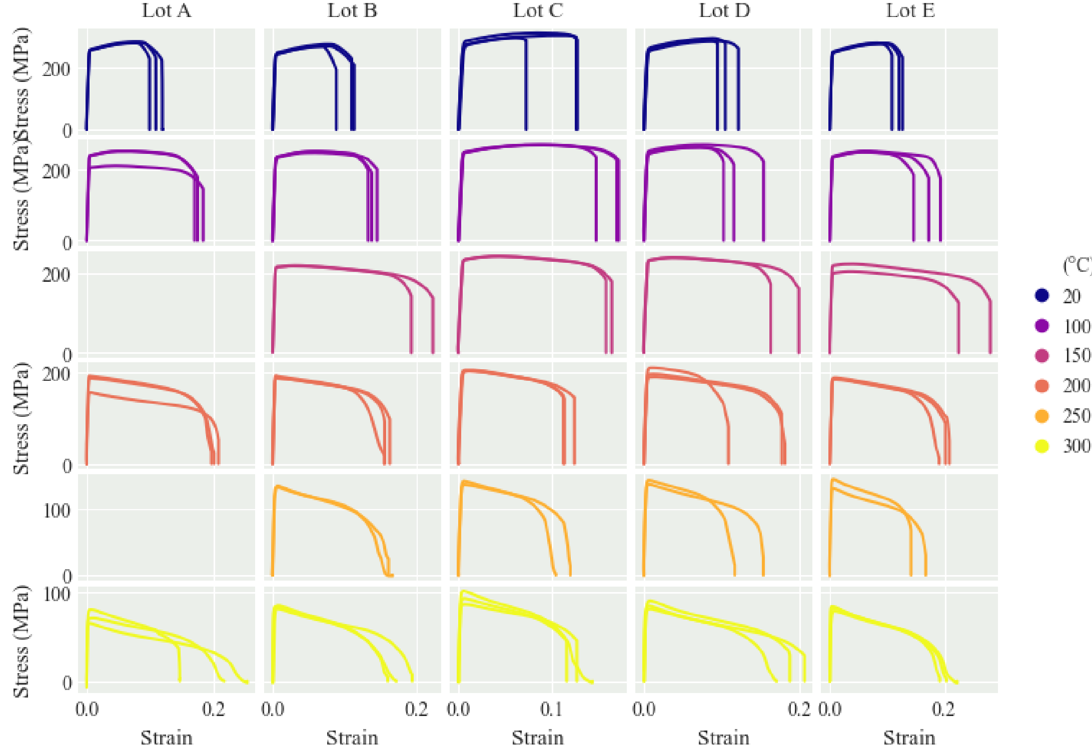
Setup the dataset plot for a single plot of all the data

[7]:
```
# define a function to plot the dataset
def ds_plot(ds, **kwargs):
    """Takes in a DataSet and extra plotting kwargs, returns a matplotlib Axes object."""
    return pam.dataset_plot(ds=ds, styler=styler, **labels_dict, **kwargs)


# plot the dataset
ds_plot(prepared_ds);
```



Setup the dataset plot for a grid of plots of the data.

[8]:
```
# define a function to plot the dataset as a grid of plots
def ds_subplots(ds, **kwargs):
    """Takes in a DataSet and extra plotting kwargs, returns an array of matplotlib Axes objects."""
    return pam.dataset_subplots(
        ds=ds, shape=(6, 5), styler=styler,
        rows_by='temperature', row_vals=[[20], [100], [150], [200], [250], [300]],
        cols_by='lot', col_vals=[['A'], ['B'], ['C'], ['D'], ['E']],
        col_titles=[f'Lot {lot}' for lot in 'ABCDE'], **labels_dict, **kwargs)


# plot the dataset as a grid of plots
ds_subplots(prepared_ds);
```



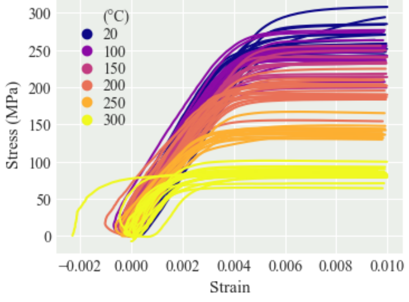
B.7.1 Processing

Find UTS and fracture point.

[9]:
```
# determine the ultimate tensile strength for each test
prepared_ds = pam.find_UTS(ds=prepared_ds, strain_key='Strain',
                           stress_key='Stress_MPa')

# determine the fracture point for each test
prepared_ds = pam.find_fracture_point(ds=prepared_ds, strain_key='Strain',
                                      stress_key='Stress_MPa')
```

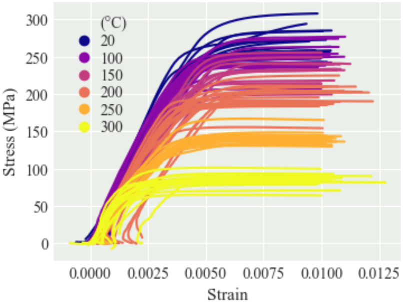
Trimming.

[10]:
```
# define a function to trim the data to a small strain range
def trim_to_small_strain(di):
    """Takes in a DataItem and returns a trimmed DataItem."""
    di.data = di.data[di.data['Strain'] < 0.01]
    return di


# apply the function to the dataset
trimmed_ds = prepared_ds.apply(trim_to_small_strain)

# plot the trimmed dataset
ds_plot(trimmed_ds);
```



Apply foot correction.

[11]:
```
# determine the upper and lower proportional limits for each test
trimmed_ds = pam.find_upl_and_lpl(ds=trimmed_ds, preload=36,
                                  preload_key='Stress_MPa')

# apply the foot correction to the dataset
corrected_ds = pam.correct_foot(ds=trimmed_ds, LPL_key='LPL', UPL_key='UPL')

# plot the corrected dataset
ds_plot(corrected_ds);
```



Make screening pdf for foot correction screening.

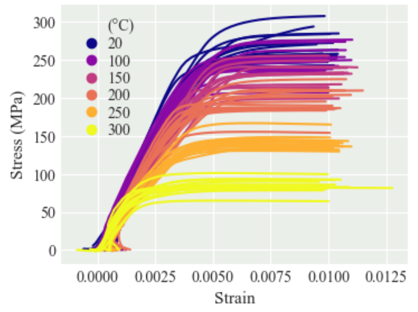
[12]:
```
# define a function to plot a single test for foot correction screening
def foot_correction_screening_plot(di):
    """Takes in a DataItem and returns a matplotlib Axes object."""
    color = styler.color_dict[di.info['temperature']]
    LPL = (di.info['UPL_0'], di.info['UPL_1'])
    UPL = (di.info['LPL_0'], di.info['LPL_1'])
    ax = ds_plot(corrected_ds.subset({'test_id': di.test_id}))
    ax = ds_plot(trimmed_ds.subset({'test_id': di.test_id}), alpha=0.5, ax=ax)
    ax.axline(UPL, slope=di.info['E'], c=color, ls='--', alpha=0.5)
    ax.plot(*UPL, c=color, marker=4)
    ax.plot(*LPL, c=color, marker=5)
    return ax


# make the screening pdf
# pam.make_screening_pdf(corrected_ds, plot_func=foot_correction_screening_plot,
#                        pdf_path='info/foot-correction screening.pdf')
```

Read screening pdf.

[13]:
```
# read the annotated screening pdf fields to the dataset
corrected_ds = pam.read_screening_pdf(ds=corrected_ds,
                                      pdf_path='info/screening-marked.pdf')

# get the metadata table of the rejected items
# corrected_ds.info_table[corrected_ds.info_table['reject'] == 'True']
```

Remove the rejected items

[14]:
```
# remove the rejected tests from the dataset
screened_ds = pam.remove_rejected_items(ds=corrected_ds, reject_key='reject')

# plot the screened dataset
ds_plot(screened_ds);
```

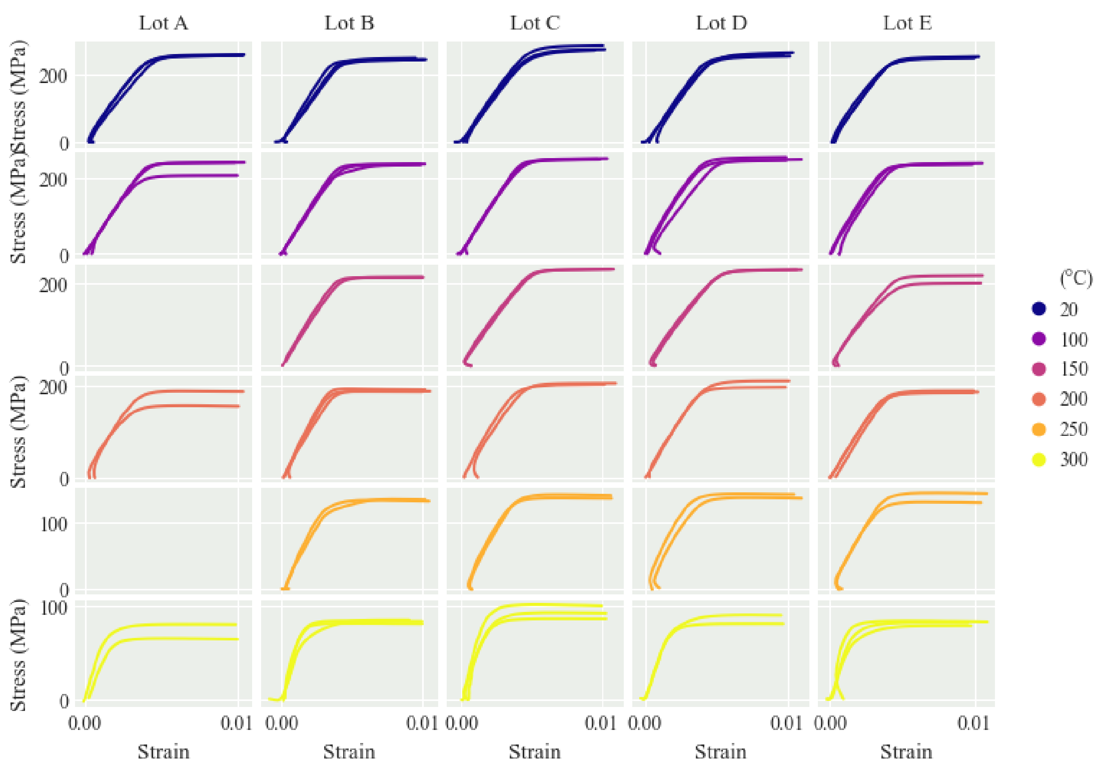


Algorithmically find the proof stresses.

[15]:
```
# find the 0.2% proof stress for each test
screened_ds = pam.find_proof_stress(ds=screened_ds, proof_strain=0.002, E_key='E',
                                    strain_key='Strain', stress_key='Stress_MPa')
```

Write processed data.

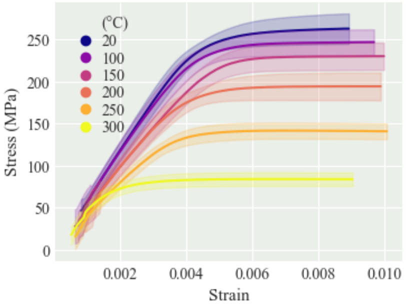
[16]:
```
# write the metadata table data files to the specified paths
screened_ds.write_output('info/02 processed info.xlsx',
                         'data/02 processed data')

# load the processed dataset
processed_ds = pam.DataSet('info/02 processed info.xlsx',
                           'data/02 processed data')

# plot the processed dataset as a grid of plots
ds_subplots(processed_ds);
```



B.7.2 Data Aggregation

Generate representative curves.

[17]:
```
# make representative curves for each temperature and test type
pam.make_representative_data(
    ds=processed_ds, info_path='info/03 repres info.xlsx',
    data_dir='data/03 repres data', repres_col='Stress_MPa',
    interp_by='Strain', interp_range='inner',
    group_by_keys=['temperature', 'test_type'],
    group_info_cols=['UTS_0', 'UTS_1', 'FP_0', 'E',
                     'PS_0.002_0', 'PS_0.002_1', 'UPL_1'])

# load the representative dataset
repres_ds = pam.DataSet('info/03 repres info.xlsx',
                        'data/03 repres data',
                        test_id_key='repres_id')

# plot representative curves with standard deviation bands
ds_plot(repres_ds,
        fill_between=('down_std_Stress_MPa', 'up_std_Stress_MPa'));

# get the metadata table of the representative dataset
# repres_ds.info_table
```

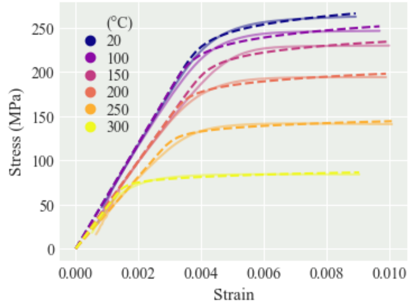


Fit models.

[18]:
```
# get the Ramberg-Osgood model function
ramberg_func = pam.models.ramberg

# set up the modelset, variables are known, params are unknown
ramberg_ms = pam.ModelSet(
    model_func=pam.models.ramberg,
    var_names=['E', 'UPL_1'],
    param_names=['H', 'n'],
    bounds=[(0., 1000.), (0.01, 0.8)]
)

# fit the modelset to the representative dataset
ramberg_ms.fit_to(repres_ds, x_key='Strain', y_key='Stress_MPa')

# get a table of the fitted parameters
# ramberg_ms.params_table
```

Predict data from fitted models.

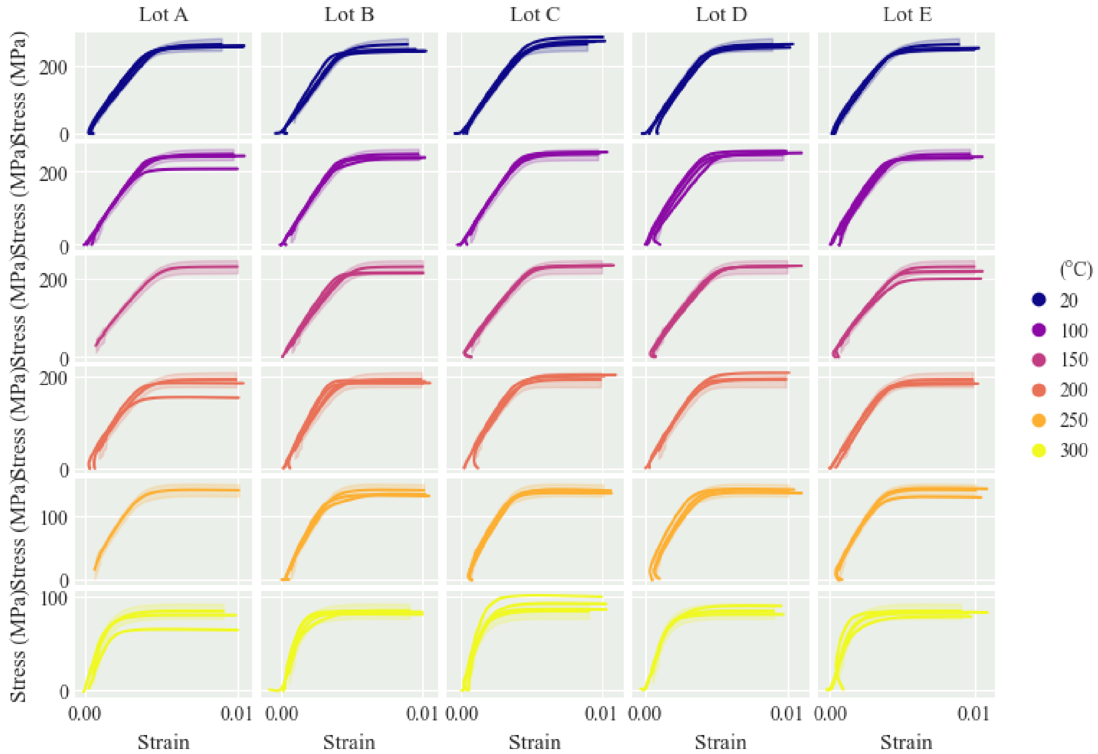
[19]:
```
# Predict a dataset from the modelset
ramberg_ds = ramberg_ms.predict()

# Plot the fitted curves above the representative curves
ax = ds_plot(repres_ds, alpha=0.5)
ds_plot(ramberg_ds, ls='--', ax=ax);
```



B.7.3 Quality Control

Check for outliers.

[20]:
```
# Plot the processed curves on a grid of subplots
axs = ds_subplots(processed_ds)

# Plot the representative curve for a row on all the plots in that row
for i, temp in enumerate(processed_ds.info_table['temperature'].unique()):
    for j in range(axs.shape[1]):
        ax = axs[i, j]
        ds_plot(ds=repres_ds.subset({'temperature': temp}),
                ax=axs[i, j], plot_legend=False,
                fill_between=('up_std_Stress_MPa', 'down_std_Stress_MPa'))
```



Read placeholders from the test report template.

[21]:
```
receipts = pam.TestReceipts(template_path=r'receipts_template.tex')
receipts.parse_placeholders(as_dict=True)
```

[21]:
```
{'table_info': None, 'plot_processed': None, 'plot_raw': None, 'test_id': None}
```

Setup functions for the test reports.

[22]:
```
import matplotlib.lines as mlines
import matplotlib.pyplot as plt

def receipts_raw_plot(di):
    test_id = di.test_id
    color = styler.color_dict[di.info['temperature']]
    UTS = (di.info['UTS_0'], di.info['UTS_1'])
    ax = ds_plot(prepared_ds.subset({'test_id': test_id}), title=f'Raw Data')
    ax.plot(*UTS, color=color, marker='s', label='UTS')
    plot_name = f'{test_id} raw.pdf'
    handles = [mlines.Line2D([], [], color=color, marker='s', ls='None', label='UTS')]
    ax.legend(handles=handles, loc='best')
    plt.savefig(plot_name, bbox_inches='tight', dpi=150)
    return plot_name


def receipts_processed_plot(di):
    test_id = di.test_id
    color = styler.color_dict[di.info['temperature']]
    UPL = (di.info['UPL_0'], di.info['UPL_1'])
    LPL = (di.info['LPL_0'], di.info['LPL_1'])
    PS_0_002 = (di.info['PS_0.002_0'], di.info['PS_0.002_1'])
    ax = ds_plot(trimmed_ds.subset({'test_id': di.test_id}), alpha=0.5)
    ds_plot(corrected_ds.subset({'test_id': di.test_id}), ax=ax, title='Processed Data')
    ax.axline(UPL, slope=di.info['E'], c=color, ls='--', alpha=0.5)
    ax.axline(PS_0_002, slope=di.info['E'], c=color, ls='--', alpha=0.5)
    ax.plot(*UPL, c=color, marker=4)
    ax.plot(*LPL, c=color, marker=5)
    ax.plot(*PS_0_002, c=color, marker='*')
    handles = [mlines.Line2D([], [], color=color, marker=4, ls='None', label='UPL'),
               mlines.Line2D([], [], color=color, marker=5, ls='None', label='LPL'),
               mlines.Line2D([], [], color=color, marker='*', ls='None', label='PS_0.002')]
    plot_name = f'{test_id} processed.pdf'
    ax.legend(handles=handles, loc='best')
    plt.savefig(plot_name, bbox_inches='tight', dpi=150)
    return plot_name


def receipts_table(di):
    series = di.info
    for key in ['UTS_1', 'UPL_1', 'LPL_1', 'PS_0.002_1', 'E']:
        series[key] = round(series[key], 2)
    for key in ['UTS_0', 'UPL_0', 'LPL_0', 'PS_0.002_0', 'FP_0']:
        series[key] = round(series[key], 4)
    # series = series.style.to_latex(escape=False, header=False)
    # table_string = series.to_frame().style.format(escape='latex').to_latex()
    table_string = series.to_frame().style.to_latex()
    return table_string
```

Generate the test reports.

[23]:
```
# Define a dictionary of functions for replacing the placeholders
replace_dict = {'test_id': lambda di: di.test_id,
                'plot_raw': receipts_raw_plot,
                'plot_processed': receipts_processed_plot,
                'table_info': receipts_table}

# Generate the combined test receipts PDF
receipts.generate_receipts(ds=processed_ds,
                           receipts_path='info/test_receipts.pdf',
                           replace_dict=replace_dict);
```

C Processing Notebooks

C.1 CS1 Processing Report

Import libraries, and print package versions.

[1]:
```
import paramaterial as pam
from paramaterial import DataSet, DataItem, ModelSet
from paramaterial.models import ramberg
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

print(pam.__version__)
print(pd.__version__)
print(sns.__version__)
print(mpl.__version__)
```

```
0.1.0
1.4.4
0.11.2
3.5.3
```

Gather data and info

Extract info from the filenames and make the info table.

[2]:
```
info_lists = [[filename] + filename.split('_')[:4] for filename in os.listdir('data/01 raw data')]
info_table = pd.DataFrame(info_lists,
                          columns=['old_filename', 'test_type', 'temperature', 'lot', 'number']
                          ).sort_values(by='test_type', ascending=False)
```

Add a unique test ID column.

[3]:
```
info_table['test_id'] = [f'test_ID_{i + 1:03d}' for i in range(len(info_table))]
info_table = info_table.set_index('test_id').reset_index()  # move the test_id column to the far left
```

Drop the PST tests. We are only going to be processing and analysing the uniaxial tests.

[4]:
```
info_table['test_type'] = info_table['test_type'].replace('T', 'UT')
info_table['test_type'] = info_table['test_type'].replace('P', 'PST')
info_table = info_table[info_table['test_type'] != 'PST']
info_table = info_table[~info_table['lot'].isin(list('FGHI'))]
```

Add the information from the paper, and convert the temperatures to numbers.

[5]:
```
info_table['rate'] = 8.66e-4  # units (/s) and all tests performed at same rate
info_table['A_0_(mm)'] = np.where(info_table['test_type'] == 'UT', 40.32, 20.16)
info_table['h_0_(mm)'] = 3.175
info_table['temperature'] = pd.to_numeric(info_table['temperature'])
```

Format the data files.

In this example, the files are already in .csv format. We just check that the column headers are the same and that there are no duplicates, then rename the files by test id.

Check column headers, uniqueness.

[6]:
```
pam.check_column_headers('data/01 raw data')
pam.check_for_duplicate_files('data/01 raw data')
```

```
Checking column headers...
First file headers:
        ['Strain', 'Stress_MPa']
Headers in all files are the same as in the first file, except for None.
Checking for duplicate files...
No duplicate files found in "data/01 raw data".
```

Write the prepared data and rename the files by test id. Also write the prepared info table.

[7]:
```
pam.copy_data_and_rename_by_test_id(data_in='data/01 raw data', data_out='data/01 prepared data',
                                    info_table=info_table)
info_table.to_excel('info/01 prepared info.xlsx', index=False)
```

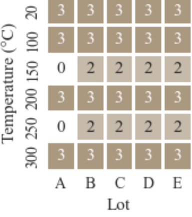
```
Copied 76 files in data/01 raw data to data/01 prepared data.
```

Make the experimental matrix

We want to identify useful groupings and make visualisations. The tests can be grouped by lot and temperature, with up to 3 repeated tests.

[8]:
```
gold_cmap = mpl.colors.LinearSegmentedColormap.from_list("", ["white", (85/255, 49/255, 0)])
mpl.rcParams["axes.facecolor"] = gold_cmap(0.1)
```

[9]:
```
plt.figure(figsize=(2, 2))
pam.experimental_matrix(info_table, index='temperature', columns='lot', as_heatmap=True, cmap=gold_cmap, xlabel='Lot',
                        ylabel='Temperature (°C)', vmax=6);
```

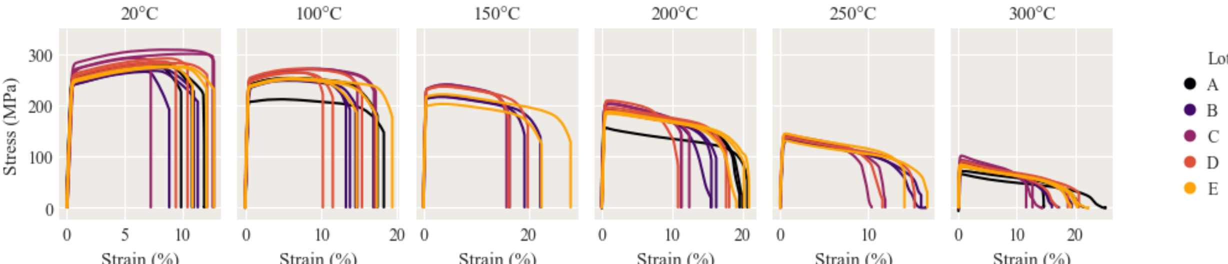


Visualise the prepared data.

[10]:
```
def make_strain_percent(di):
    di.data['Strain'] = di.data['Strain']*100
    return di


prepared_ds = DataSet('info/01 prepared info.xlsx', 'data/01 prepared data').sort_by(
    ['temperature', 'lot']).apply(make_strain_percent)
```

[11]:
```
lot_styler = pam.Styler(color_by='lot', color_by_label='Lot', cmap='inferno').style_to(prepared_ds)


def ds_subplots(ds: DataSet, **kwargs):
    temperatures = sorted(prepared_ds.info_table['temperature'].unique())
    return pam.dataset_subplots(
        ds=ds, x='Strain', y='Stress_MPa', xlabel='Strain (%)', ylabel='Stress (MPa)',
        styler=lot_styler, plot_legend=False, figsize=(12, 2), shape=(1, 6), ylim=(-25., 350.),
        wspace=0.1, rows_by='test_type', cols_by='temperature', row_vals=[['UT']],
        col_vals=[[T] for T in temperatures], col_titles=[f'{T}°C' for T in temperatures],
        **kwargs)
```

```
ds_subplots(prepared_ds);
```



Find UTS and Failure

[12]:
```
def find_uts_and_failure(di: DataItem):
    di.info['UTS_1'] = di.data['Stress_MPa'].max()
    di.info['UTS_0'] = di.data['Strain'][di.data['Stress_MPa'].idxmax()]
    di.info['FP_0'] = di.data['Strain'].max()
    return di


prepared_ds = prepared_ds.apply(find_uts_and_failure)
```

Trimming

[13]:
```
def trim_to_small_strain(di: DataItem):
    di.data = di.data[di.data['Strain'] < 1]
    return di


trimmed_ds = prepared_ds.apply(trim_to_small_strain)
```

Foot Correction

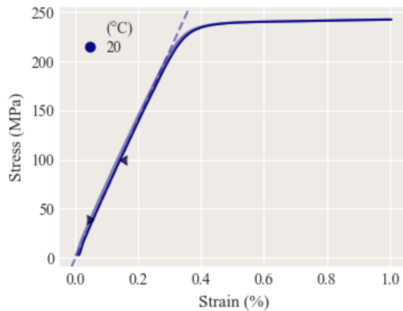
[14]:
```
corrected_ds = pam.find_upl_and_lpl(trimmed_ds, preload=36, preload_key='Stress_MPa')
corrected_ds = pam.correct_foot(corrected_ds)
```

Foot correction screening

Make screening pdf.

[15]:
```
temp_styler = pam.Styler(color_by='temperature', color_by_label='(°C)').style_to(prepared_ds)


def ds_plot(ds: DataSet, **kwargs):
    return pam.dataset_plot(ds, x='Strain', y='Stress_MPa', xlabel='Strain (%)', ylabel='Stress (MPa)',
                            styler=temp_styler, **kwargs)


def foot_correction_screening_plot(di):
    test_id = di.test_id
    temp = di.info['temperature']
    color = temp_styler.color_dict[temp]
    UPL = (di.info['UPL_0'], di.info['UPL_1'])
    LPL = (di.info['LPL_0'], di.info['LPL_1'])
    _ax = ds_plot(corrected_ds.subset({'test_id': [test_id]}))
    _ax = ds_plot(trimmed_ds.subset({'test_id': [test_id]}), alpha=0.5, ax=_ax)
    _ax.axline(UPL, slope=di.info['E'], c=color, ls='--', alpha=0.5, zorder=500 + temp)
    _ax.plot(*UPL, c='k', mfc=color, marker=4, alpha=0.8, markersize=6, zorder=1000 + temp)
    _ax.plot(*LPL, c='k', mfc=color, marker=5, alpha=0.8, markersize=6, zorder=1000 + temp)
```

```
foot_correction_screening_plot(corrected_ds[3])
pam.make_screening_pdf(corrected_ds, foot_correction_screening_plot,
                       'info/foot correction screening.pdf');
```



Reject flagged tests.

[16]:
```
rejected_ds = pam.read_screening_pdf(corrected_ds, 'info/foot correction screening marked.pdf')
rejected_items = rejected_ds.info_table[rejected_ds.info_table['reject'] == 'True']
rejected_items[['test_id', 'temperature', 'lot', 'number', 'reject', 'comment']]
```

[16]:

	test_id	temperature	lot	number	reject	comment
38	test_ID_002	200	A	3	True	
45	test_ID_015	200	C	2	True	
48	test_ID_018	200	D	2	True	
61	test_ID_041	300	A	2	True	
71	test_ID_049	300	D	2	True	

[17]:
```
screened_ds = pam.remove_rejected_items(rejected_ds)
```

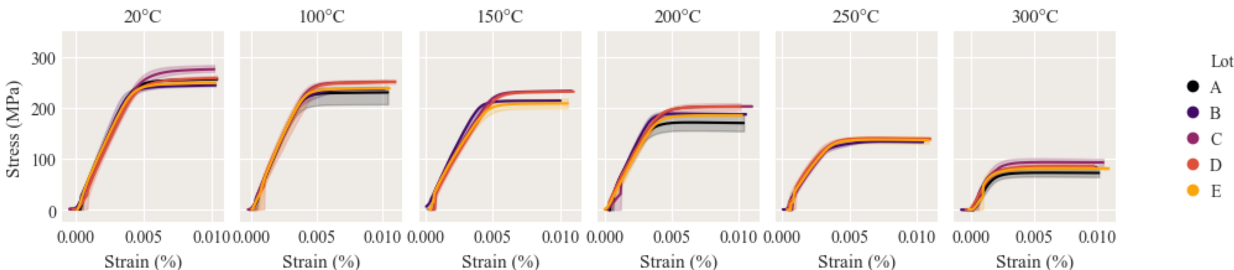
Proof Stress

[18]:
```
def reset_strain(di):
    di.data['Strain'] = di.data['Strain']/100
    di.info['E'] = di.info['E']*100
    return di


proof_ds = pam.find_proof_stress(screened_ds.apply(reset_strain))
```

[19]:
```
proof_ds.write_output('info/02 processed info.xlsx', 'data/02 processed data')
processed_ds = DataSet('info/02 processed info.xlsx', 'data/02 processed data')
```

Representative curves

[20]:
```
pam.make_representative_data(processed_ds, 'info/03 repres info.xlsx', 'data/03 repres data',
                             repres_col='Stress_MPa', group_by_keys=['lot', 'temperature', 'test_type'],
                             interp_by='Strain', group_info_cols=['E', 'PS_0.002_1', 'UTS_1', 'UTS_0', 'FP_0', 'UPL_1'])
repres_ds = DataSet('info/03 repres info.xlsx', 'data/03 repres data', test_id_key='repres_id')
ds_subplots(repres_ds, fill_between=('min_Stress_MPa', 'max_Stress_MPa'));
```



Mechanical properties

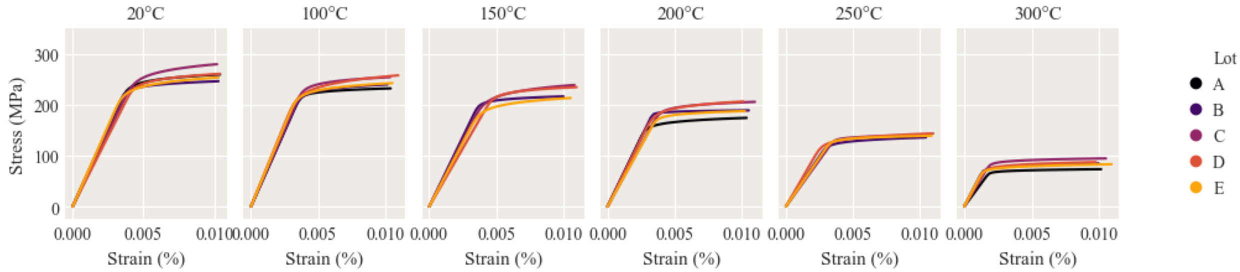
[21]:
```
table = pam.make_representative_info(processed_ds, group_by_keys=['temperature', 'test_type'],
                                     group_info_cols=['E', 'PS_0.002_1', 'UTS_1', 'UTS_0', 'FP_0', 'UPL_1'])
table[['temperature', 'nr averaged', 'E', 'PS_0.002_1', 'UTS_1', 'UTS_0', 'FP_0',  'std_PS_0.002_1', 'std_UTS_1',
      'std_UTS_0', 'std_FP_0']].style.hide(axis='index').format('{:.3g}').to_latex('info/04 mechanical properties.tex')
```

temperature	nr	E	PS_0.002_1	UTS_1	UTS_0	FP_0	std_PS_0.002_1	std_UTS_1	std_UTS_0	std_FP_0
20	15	5.98e+04	253	282	8.25	10.9	11.2	12	1.25	1.57
100	15	6.13e+04	239	256	6.57	15.4	11.3	15.1	1.31	2.53
150	8	5.06e+04	220	227	3.81	20.1	12.1	14.2	0.548	4.16
200	12	5.16e+04	188	190	0.853	16.7	13.2	13.6	0.311	3.66
250	8	4e+04	136	138	0.616	14.1	5.43	4.91	0.0567	2.48
300	13	4.65e+04	82.4	83.8	0.683	17.6	8.28	8.37	0.171	3.84

Fitted curves

[22]:
```
ramberg_ms = ModelSet(ramberg, param_names=['C', 'n'], var_names=['E', 'UPL_1'],
                      bounds=[(0., 1000.), (0.01, 0.8)], scipy_func='minimize')

ramberg_ms.fit_to(repres_ds, 'Strain', 'Stress_MPa', sample_size=40)
```

[23]:
```
ramberg_ds = ramberg_ms.predict()
ds_subplots(ramberg_ds);
```



Fitting results

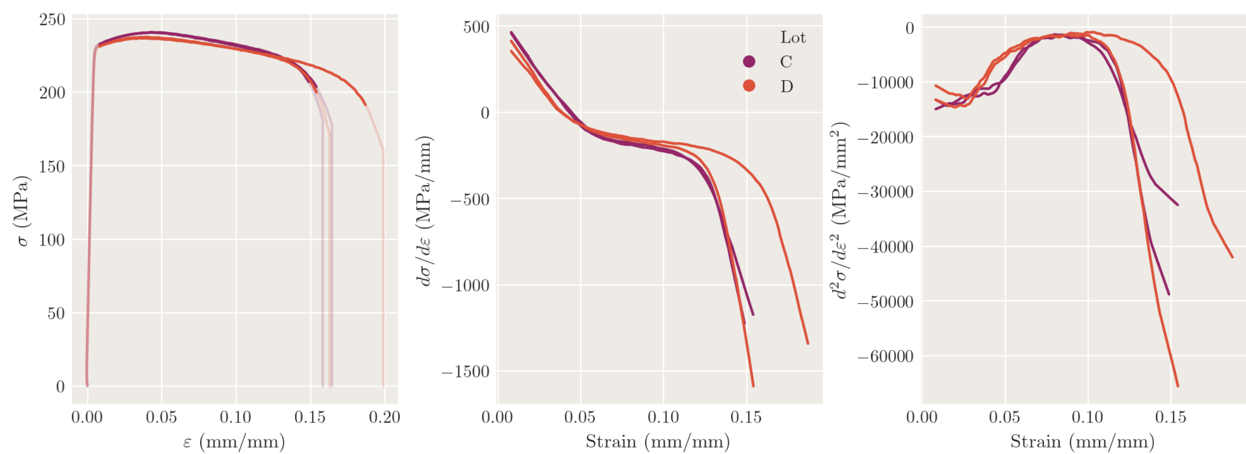
[24]:
```
table = pam.make_representative_info(ramberg_ds, group_by_keys=['temperature', 'test_type'],
                                     group_info_cols=['E',  'UPL_1', 'C', 'n', 'error'])
table[['temperature', 'nr averaged', 'E', 'UPL_1', 'C', 'n',
       'error']].style.hide(axis='index').format('{:.3g}').to_latex('info/05 fitting results.tex')
```

temperature	nr	E	UPL_1	C	n	error
20	5	5.98e+04	162	169	0.105	17
100	5	6.13e+04	136	170	0.0874	21.2
200	5	5.17e+04	118	112	0.0782	19.3
300	5	4.58e+04	51.7	57	0.115	9.08
150	4	5.06e+04	115	168	0.0792	30.7
250	4	4e+04	87.3	84.9	0.0967	18.5

Large strain analysis example

While this analysis was not included in the results of the case study presented in Section 5.1, it does provide a useful example of how Paramaterial can be used to analyse large strain data. A trimming function is used to isolate a section of the curve in the plastic region, before the first and second derivatives are determined and plotted below. These could be used by material scientist analysts to determine mechanical properties, such as the work-hardening and work softening rates or the critical strain for dynamic softening.

[194]:
```
ds = prepared_ds.subset({'temperature': [150], 'lot': ['C', 'D']})


def trim_large(di: DataItem):
        di.data = di.data[di.data['Strain'] > 0.01]
        di.data = di.data.iloc[:int(len(di.data) * 0.95)]
        return di


def find_derivatives(di: DataItem):
        from scipy.signal import savgol_filter
        di.data['Stress_MPa'] = savgol_filter(di.data['Stress_MPa'], window_length=40, polyorder=1)
        di.data['dStress_MPa/dStrain'] = np.gradient(di.data['Stress_MPa'], di.data['Strain'])
        di.data['dStress_MPa/dStrain'] = savgol_filter(di.data['dStress_MPa/dStrain'], window_length=40, polyorder=1)
        di.data['d2Stress_MPa/dStrain2'] = np.gradient(di.data['dStress_MPa/dStrain'], di.data['Strain'])
        di.data['d2Stress_MPa/dStrain2'] = savgol_filter(di.data['d2Stress_MPa/dStrain2'], window_length=40, polyorder=1)
        return di


fig, axs = plt.subplots(1, 3, figsize=(12, 4), gridspec_kw={'width_ratios': [1, 1, 1], 'wspace': 0.3})

pam.dataset_plot(ds, styler=lot_styler, x='Strain', y='Stress_MPa', xlabel='$\\varepsilon$ (mm/mm)',
                 ylabel='$\\sigma$ (MPa)', ax=axs[0], plot_legend=False, alpha=0.2)
pam.dataset_plot(ds.apply(trim_large), styler=lot_styler, x='Strain', y='Stress_MPa', xlabel='$\\varepsilon$ (mm/mm)',
                 ylabel='$\\sigma$ (MPa)', ax=axs[0], plot_legend=False)

ds = ds.apply(trim_large).apply(find_derivatives)

pam.dataset_plot(ds, styler=lot_styler, x='Strain', y='dStress_MPa/dStrain', xlabel='Strain (mm/mm)',
                 ylabel='$d\\sigma/d\\varepsilon$ (MPa/mm)', ax=axs[1])

pam.dataset_plot(ds, styler=lot_styler, x='Strain', y='d2Stress_MPa/dStrain2', xlabel='Strain (mm/mm)',
                 ylabel='$d^2\\sigma/d\\varepsilon^2$ (MPa/mm$^2$)', ax=axs[2], plot_legend=False)
```



C.2 CS2 Processing Report

Package versions.

[1]:
```
import paramaterial as pam
from paramaterial import DataSet, DataItem, ModelSet
from paramaterial.models import ramberg
import os
import pandas as pd
import seaborn as sns
import matplotlib as mpl
from matplotlib import pyplot as plt

print(pam.__version__)
print(pd.__version__)
print(sns.__version__)
print(mpl.__version__)
```

```
0.1.0
1.4.4
0.11.2
3.5.3
```

Extract info from the filenames and make the info table.

[2]:
```
info_lists = [[filename] + filename.split('_')[:4] for filename in os.listdir('data/01 raw data')]
info_table = pd.DataFrame(info_lists,
                          columns=['old_filename', 'test_type', 'temperature', 'lot', 'number']
                          ).sort_values(by='test_type', ascending=False)
```

Add a unique test ID column.

[3]:
```
info_table['test_id'] = [f'test_ID_{i + 1:03d}' for i in range(len(info_table))]
info_table = info_table.set_index('test_id').reset_index()  # move the test_id column to the far left
```

Rename test types and get subset of lots.

[4]:
```
info_table['test_type'] = info_table['test_type'].replace('T', 'UT')
info_table['test_type'] = info_table['test_type'].replace('P', 'PST')
info_table = info_table[info_table['lot'].isin(list('FGHI'))]
```

Add the information from the paper, and convert the temperatures to numbers.

[5]:
```
info_table['rate'] = 8.66e-4  # units (/s)
info_table['temperature'] = pd.to_numeric(info_table['temperature'])
info_table = info_table[info_table['temperature'] != 100]  #no PST at 100
```

Format the data files.

In this example, the files are already in .csv format. We just check that the column headers are the same and that there are no duplicates, then rename the files by test id.

Check column headers, uniqueness.

[6]:
```
pam.preparing.check_column_headers('data/01 raw data')
pam.preparing.check_for_duplicate_files('data/01 raw data')
```

```
Checking column headers...
First file headers:
        ['Strain', 'Stress_MPa']
Headers in all files are the same as in the first file, except for None.
Checking for duplicate files...
No duplicate files found in "data/01 raw data".
```

Write the prepared data and rename the files by test id. Also write the prepared info table.

[7]:
```
pam.preparing.copy_data_and_rename_by_test_id(data_in='data/01 raw data', data_out='data/01 prepared data',
                                              info_table=info_table)
info_table.to_excel('info/01 prepared info.xlsx', index=False)
```

```
Copied 74 files in data/01 raw data to data/01 prepared data.
```

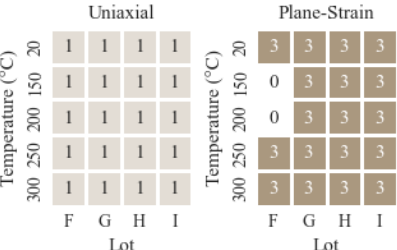
[8]:
```
def make_strain_percent(di):
    di.data['Strain'] = di.data['Strain']*100
    return di


prepared_ds = DataSet('info/01 prepared info.xlsx', 'data/01 prepared data').sort_by(
    ['test_type', 'temperature', 'lot']).apply(make_strain_percent)
```

Make the experimental matrix

The tests can be grouped by lot and temperature, with up to 3 repeated tests.

[9]:
```
gold_cmap = mpl.colors.LinearSegmentedColormap.from_list("", ["white", (85/255, 49/255, 0)])
mpl.rcParams["axes.facecolor"] = gold_cmap(0.1)
```

[10]:
```
fig, (ut_ax, pst_ax) = plt.subplots(1, 2, figsize=(3.8, 1.8))

pam.experimental_matrix(prepared_ds.subset({'test_type': 'UT'}).info_table, cmap=gold_cmap, ax=ut_ax, vmin=0, vmax=6,
                        index='temperature', columns='lot', as_heatmap=True, xlabel='Lot', ylabel='Temperature (°C)',
                        title='Uniaxial')

pam.experimental_matrix(prepared_ds.subset({'test_type': 'PST'}).info_table, cmap=gold_cmap, ax=pst_ax, vmin=0, vmax=6,
                        index='temperature', columns='lot', as_heatmap=True, xlabel='Lot', ylabel='Temperature (°C)',
                        title='Plane-Strain');
```

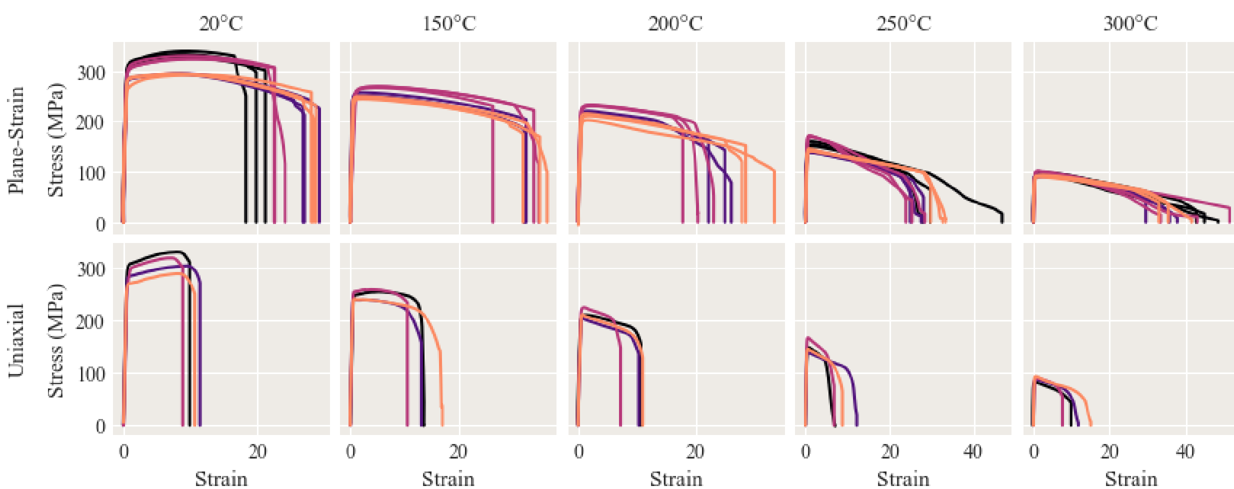


Visualise the prepared data.

We could colour by lot or by temperature. Colouring by lot is useful to observe variation, colouring by temperature is useful to observe temperature variation.

[11]:
```
lot_styler = pam.Styler(color_by='lot', color_by_label='Lot', cmap='magma').style_to(prepared_ds)


def ds_subplots(ds: DataSet, **kwargs):
    temperatures = sorted(prepared_ds.info_table['temperature'].unique())
    return pam.plotting.dataset_subplots(
        ds=ds, x='Strain', y='Stress_MPa', xlabel='Strain', ylabel='Stress (MPa)',
        styler=lot_styler.style_to(prepared_ds), plot_legend=False,
        rows_by='test_type', row_vals=[['PST'], ['UT']], row_titles=['Plane-Strain', 'Uniaxial'],
        figsize=(10, 3.5), shape=(2, 5), cols_by='temperature',
        col_vals=[[T] for T in temperatures], col_titles=[f'{T}°C' for T in temperatures], **kwargs,
```

```
        subplot_legend=False)

ds_subplots(prepared_ds);
```

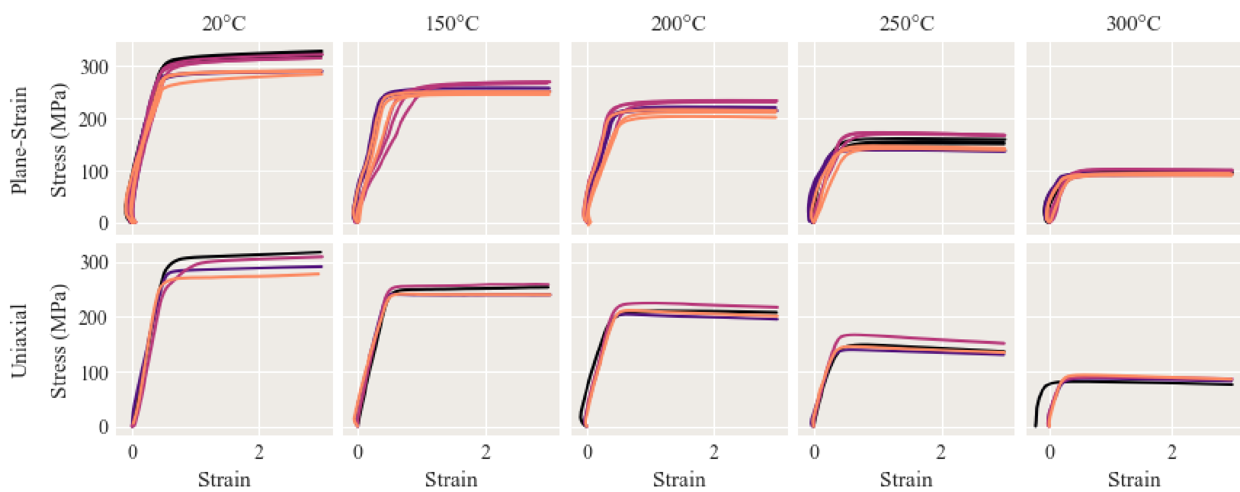


UTS, failure point

[12]:
```
def find_uts(di: DataItem):
    di.info['UTS_0'] = di.data['Strain'].iloc[di.data['Stress_MPa'].idxmax()]
    di.info['UTS_1'] = di.data['Stress_MPa'].max()
    return di


def find_fracture(di: DataItem):
    di.info['FP_0'] = di.data['Strain'].max()
    return di


props_ds = prepared_ds.apply(find_uts).apply(find_fracture)
```

Trimming

[13]:
```
def trim_to_small_strain(di: DataItem):
    di.data = di.data[di.data['Strain'] < 3]
    return di


trimmed_ds = props_ds.apply(trim_to_small_strain)
ds_subplots(trimmed_ds);
```



Foot correction, Young's modulus

[14]:
```
corrected_ds = pam.find_upl_and_lpl(trimmed_ds, preload=36, preload_key='Stress_MPa')
corrected_ds = pam.correct_foot(corrected_ds)
```

[15]:
```
corrected_ds.write_output('info/02 processed info.xlsx', 'data/02 processed data')
processed_ds = DataSet('info/02 processed info.xlsx', 'data/02 processed data')
```

Mechanical Properties

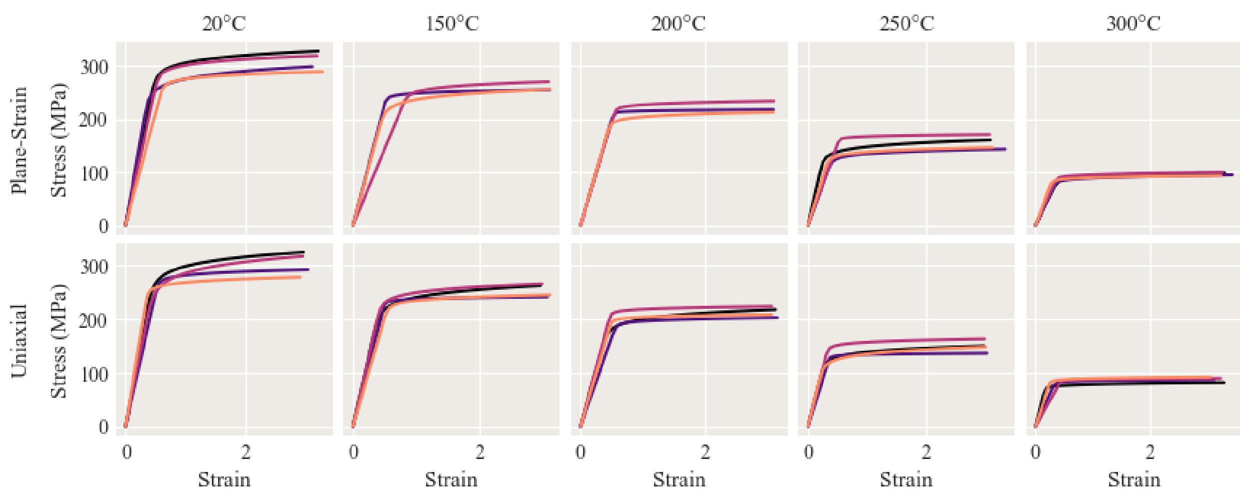
[16]:
```
table = pam.make_representative_info(ds=processed_ds, group_by_keys=['temperature', 'test_type'],
                                     group_info_cols=['E', 'UPL_0', 'UPL_1', 'UTS_0', 'UTS_1', 'FP_0'])
table = table[
    ['temperature', 'test_type', 'E', 'UPL_0', 'UPL_1', 'UTS_0', 'UTS_1', 'FP_0', 'std_E', 'std_UPL_0', 'std_UPL_1',
     'std_UTS_0', 'std_UTS_1', 'std_FP_0']].style.hide(axis='index').to_latex('info/04 properties.tex')
```

temp	test	nr	E	UPL_0	UPL_1	UTS_0	UTS_1	FP_0	std_E	std_UPL_0	std_UPL_1	std_UTS_0	std_UTS_1	std_FP_0
20	PST	12	543	0.409	194	9	312	24.7	132	0.243	72.3	1.23	19.2	3.78
20	UT	4	589	0.293	169	8.15	311	10.2	80.1	0.118	56.9	0.968	18	1.09
150	PST	9	394	0.516	187	3.15	256	32.6	98.2	0.265	67.6	1.49	10	2.86
150	UT	4	477	0.352	163	2.62	249	13.5	57.5	0.125	45.2	2.33	9.4	2.66
200	PST	9	396	0.478	185	1.83	220	25	48.8	0.123	42.4	0.368	10.4	4.71
200	UT	4	396	0.403	157	0.936	213	9.8	46.7	0.119	41.3	0.359	8.56	1.73
250	PST	12	367	0.296	93.3	1.3	152	29.8	107	0.191	41.7	0.367	12.8	6.03
250	UT	4	417	0.21	84.5	0.653	150	8.69	46.2	0.114	38.6	0.0803	11.8	2.5
300	PST	12	248	0.323	68.8	1.92	95	39.8	83.5	0.173	17.3	0.81	4.04	6.72
300	UT	4	303	0.258	68.5	0.626	88.3	11.1	110	0.119	9.45	0.158	4.73	3.17

Fitted model curves

[17]:
```
pam.make_representative_data(processed_ds, 'info/03 repres info.xlsx', 'data/03 repres data',
                             repres_col='Stress_MPa', group_by_keys=['lot', 'temperature', 'test_type', 'rate'],
                             interp_by='Strain', group_info_cols=['E', 'UPL_1'])
repr_ds = DataSet('info/03 repres info.xlsx', 'data/03 repres data', test_id_key='repres_id')

ramberg_ms = ModelSet(ramberg, param_names=['C', 'n'], var_names=['E', 'UPL_1'],
                      bounds=[(0., 1000.), (0.1, 0.8)], scipy_func='minimize')

ramberg_ms.fit_to(repr_ds, 'Strain', 'Stress_MPa', sample_size=40)
ramberg_ds = ramberg_ms.predict(xmin=0, xmax=2.2)
```

```
ds_subplots(ramberg_ds);
```



Fitting results

```
[18]: table = pam.make_representative_info(ramberg_ds, group_by_keys=['temperature', 'test_type'],
                                            group_info_cols=['E', 'UPL_1', 'C', 'n', 'error'])
      table[['test_type', 'temperature', 'nr averaged', 'E', 'UPL_1', 'C', 'n',
             'error']].style.hide(axis='index').to_latex('info/05 fitting results.tex')
```

test_type	temperature	nr	E	UPL_1	C	n	error
PST	20	4	543	194	103	0.11	79.4
UT	20	4	589	169	123	0.1	28.2
PST	150	3	394	187	66.3	0.116	25.5
UT	150	4	477	163	83.1	0.1	41.4
PST	200	3	396	185	32.4	0.112	23.7
UT	200	4	396	157	51.2	0.1	78.5
PST	250	4	367	93.3	55.8	0.1	64.5
UT	250	4	417	84.5	58.9	0.1	72.9
PST	300	4	248	68.8	24.1	0.157	26.4
UT	300	4	303	68.5	17.1	0.1	45.8

C.3 CS3 Processing Report

This example showcases the use of Paramaterial to process and analyse a set of uniaxial compression test data.

[1]:
```
import paramaterial as pam
import numpy as np
from matplotlib import pyplot as plt
import matplotlib as mpl
from paramaterial import DataSet, DataItem
print(pam.__version__)
print(np.__version__)
print(mpl.__version__)
```

```
0.1.0
1.21.6
3.5.3
```

Data preparation

[2]:
```
pam.check_column_headers('data/01 raw data')
pam.check_for_duplicate_files('data/01 raw data')
```

```
Checking column headers...
First file headers:
        ['Time(sec)', 'Force(kN)', 'Jaw(mm)', 'PowAngle(deg)', 'Power(W)',
'Pram', 'PTemp', 'Strain', 'Stress(MPa)', 'Stroke(mm)', 'TC1(C)', 'wedge(mm)']
Headers in all files are the same as in the first file, except for None.
Checking for duplicate files...
No duplicate files found in "data/01 raw data".
```

Load the data and info into a dataset object (ds is shorthand for dataset).

[3]:
```
raw_ds = DataSet('info/01 raw info.xlsx', 'data/01 raw data').sort_by('temperature')
```

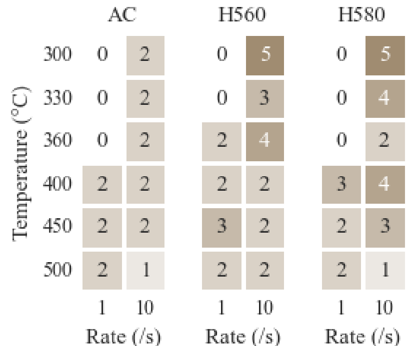
Make the experimental matrix

We want to identify useful groupings and make visualisations.

The tests can be grouped by nominal rate, nominal temperature, and material. We will make the classic rate-temperature matrices - one for each material.

[4]:
```
cmap_gold = mpl.colors.LinearSegmentedColormap.from_list("", ["white", (85/255, 49/255, 0)])
mpl.rcParams['axes.facecolor'] = cmap_gold(0.1)
```

[5]:
```
fig, axs = plt.subplots(1, 3, figsize=(3, 2.3))
heatmap_kwargs = dict(linewidths=2, cbar=False, annot=True, cmap=cmap_gold, vmax=9)

pam.experimental_matrix(raw_ds.subset({'material': 'AC'}).info_table,
                        index='temperature', columns='rate', as_heatmap=True,
                        title='AC', xlabel='Rate (/s)', ylabel='Temperature (°C)',
                        tick_params=dict(rotation=0), ax=axs[0], **heatmap_kwargs)

pam.experimental_matrix(raw_ds.subset({'material': 'H560'}).info_table,
                        index='temperature', columns='rate', as_heatmap=True,
                        title='H560', xlabel='Rate (/s)', ylabel=' ',
                        tick_params=dict(labelleft=False), ax=axs[1], **heatmap_kwargs)

pam.experimental_matrix(raw_ds.subset({'material': 'H580'}).info_table,
                        index='temperature', columns='rate', as_heatmap=True,
                        title='H580', xlabel='Rate (/s)', ylabel=' ',
                        tick_params=dict(labelleft=False), ax=axs[2], **heatmap_kwargs);
```



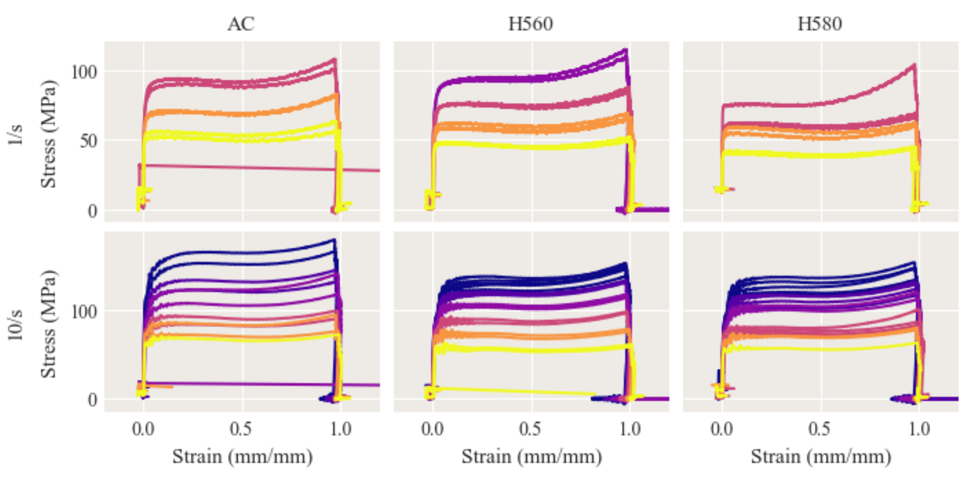
Setup data overview plot

Now that we know how to group the data, we can set up a plotting function.

[6]:
```
styler = pam.plotting.Styler(color_by='temperature', color_by_label='(°C)', cmap='plasma').style_to(raw_ds)


def ds_subplot(ds: DataSet, **kwargs):
    return pam.plotting.dataset_subplots(
        ds, shape=(2, 3), xlim=(-0.2, 1.2), figsize=(8, 3.5),
        rows_by='rate', cols_by='material',
        row_vals=[[1], [10]], col_vals=[['AC'], ['H560'], ['H580']],
        col_titles=['AC', 'H560', 'H580'], row_titles=['1/s','10/s'],
        styler=styler, subplot_legend=False, plot_legend=False,
        **kwargs
    )


stress_strain_labels = dict(x='Strain', y='Stress(MPa)', xlabel='Strain (mm/mm)', ylabel='Stress (MPa)')
```

Plot the raw data.

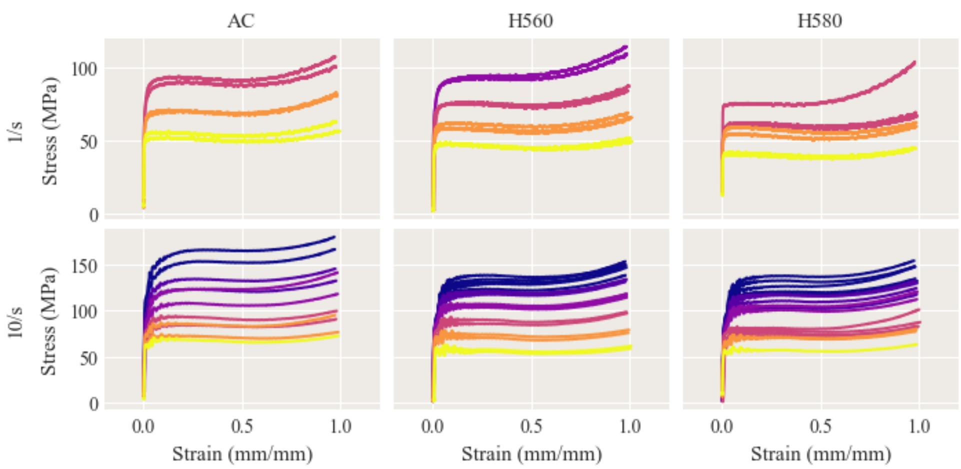
[7]:
```
ds_subplot(raw_ds, **stress_strain_labels);
```



Trim data

[8]:
```
def drop_columns(di: DataItem) -> DataItem:
    di.data = di.data.drop(columns=['PowAngle(deg)', 'Power(W)', 'Pram', 'PTemp', 'Stroke(mm)', 'wedge(mm)'])
    return di


def trim_using_time_step(di: DataItem) -> DataItem:
    t_diff = np.diff(di.data['Time(sec)'])
    di.data['time diff'] = np.hstack([t_diff[0], t_diff])
    di.data = di.data[di.data['time diff'] < 0.02][1:]
    return di


def remove_trailing_data(di: DataItem):
    di.data = di.data.iloc[:di.data['Force(kN)'].idxmax()]
    return di
```

[9]:
```
trimmed_ds = raw_ds.apply(drop_columns).apply(trim_using_time_step).apply(remove_trailing_data)
ds_subplot(trimmed_ds, **stress_strain_labels);
```



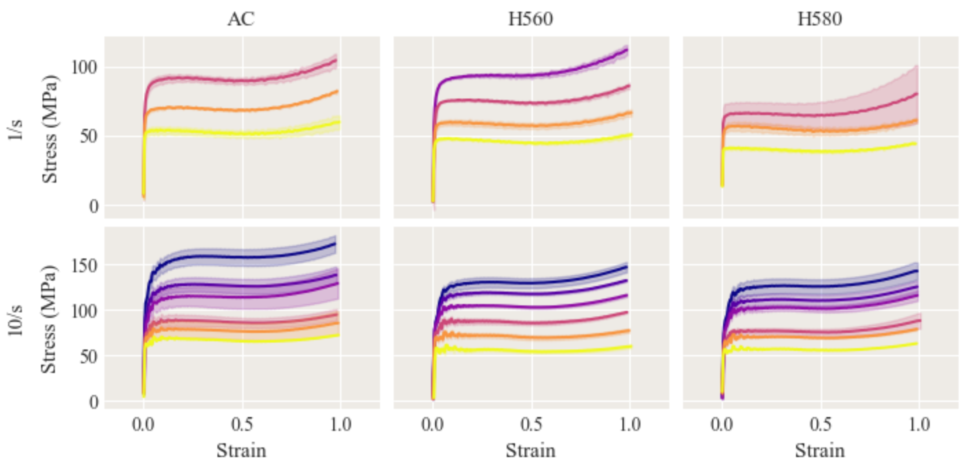
Find flow stresses

Determine the stress, temperature for each test. Note that while nominal temperature is given, the actual test conditions are slightly different. The actual strain-rate does not differ significantly from the nominal rate for these tests.

[10]:
```
rates_ds = pam.calculate_strain_rate(trimmed_ds, time_key='Time(sec)')
processed_ds = pam.find_flow_stress_values(rates_ds, flow_strain=0.3,
                                            stress_key='Stress(MPa)', temperature_key='TC1(C)', rate_key='Strain_Rate')

processed_ds.write_output('info/02 processed info.xlsx', 'data/02 processed data')
```

Make representative curves

[11]:
```
pam.make_representative_data(processed_ds, 'info/03 repr trim info.xlsx', 'data/03 repr trim data',
                             repres_col='Stress(MPa)', group_by_keys=['temperature', 'material', 'rate'],
                             interp_by='Strain', group_info_cols=['flow_Stress(MPa)'])
repr_ds = DataSet('info/03 repr trim info.xlsx', 'data/03 repr trim data', test_id_key='repres_id').sort_by('temperature')
ds_subplot(repr_ds, x='Strain', y='Stress(MPa)', xlabel='Strain', ylabel='Stress (MPa)',
           fill_between=('down_std_Stress(MPa)', 'up_std_Stress(MPa)'));
```



Zener-Holloman Model

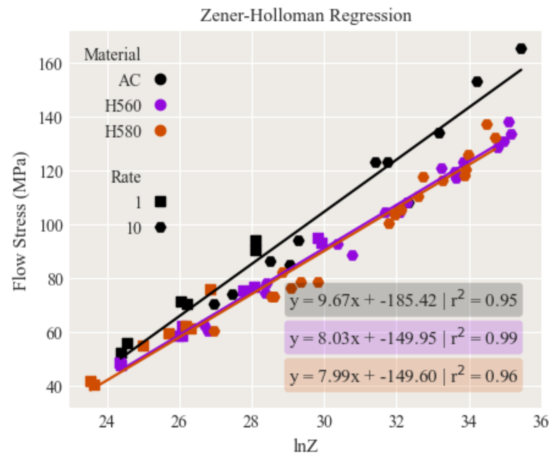
[12]:
```
processed_ds.info_table['Q_activation'] = 155000.
processed_ds.info_table['flow_temp(K)'] = processed_ds.info_table['flow_TC1(C)'] + 273.15
```

[13]:
```
analysis_ds = processed_ds.apply(pam.calculate_ZH_parameter, rate_key='flow_Strain_Rate',
                                 temperature_key='flow_temp(K)').sort_by(['material', 'rate'])
```

[14]:
```
from matplotlib.colors import ListedColormap
default_cmap = plt.cm.get_cmap('RdYlGn')
colors = default_cmap(np.arange(default_cmap.N))
colors[0] = (1, 1, 1, 1)
custom_cmap = ListedColormap(colors)


def conformance_matrix(_ds: DataSet, axs):
    heatmap_kwargs = dict(linewidths=4, cbar=False, annot=True, fmt='.1f', cmap=custom_cmap, vmin=80, vmax=100)
    pam.make_quality_matrix(_ds.subset({'material': 'AC'}).info_table, index='temperature', columns='rate',
                            flow_stress_key='flow_Stress(MPa)', as_heatmap=True, title='AC',
                            xlabel='Rate (/s)', ylabel='Temperature (°C)', tick_params=dict(rotation=0), ax=axs[0],
                            **heatmap_kwargs)
    pam.make_quality_matrix(_ds.subset({'material': 'H560'}).info_table, index='temperature', columns='rate',
                            flow_stress_key='flow_Stress(MPa)', as_heatmap=True, title='H560',
                            xlabel='Rate (/s)', ylabel=' ', tick_params=dict(labelleft=False), ax=axs[1],
                            **heatmap_kwargs)
    pam.make_quality_matrix(_ds.subset({'material': 'H580'}).info_table, index='temperature', columns='rate',
                            flow_stress_key='flow_Stress(MPa)', as_heatmap=True, title='H580',
                            xlabel='Rate (/s)', ylabel=' ', tick_params=dict(labelleft=False), ax=axs[2],
                            **heatmap_kwargs)
    return axs
```

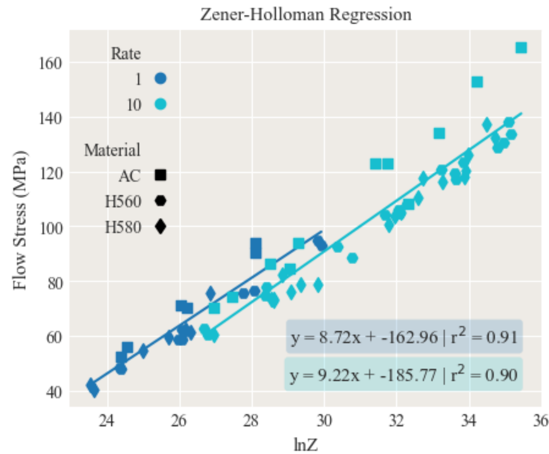
[15]:
```
material_analysis_ds = pam.apply_ZH_regression(analysis_ds.copy(), flow_stress_key='flow_Stress(MPa)',
                                               group_by='material')
rate_analysis_ds = pam.apply_ZH_regression(analysis_ds, flow_stress_key='flow_Stress(MPa)', group_by='rate')
```

[16]:
```
fig, axs = plt.subplots(1, 1, figsize=(5, 4))
mat_cmap = mpl.colors.LinearSegmentedColormap.from_list("", mpl.cm.gnuplot(np.linspace(0, 1, 4))[:4])

pam.plot_ZH_regression(material_analysis_ds, group_by='material', color_by='material', cmap=mat_cmap, marker_by='rate',
                       flow_stress_key='flow_Stress(MPa)', rate_key='rate', temperature_key='flow_temp(K)', ax=axs,
                       eq_hscale=0.1);
```



[17]:
```
fig, axs = plt.subplots(1, 3, figsize=(3.5, 4))
conformance_matrix(material_analysis_ds, axs);
```

[18]:
```
fig, axs = plt.subplots(1, 1, figsize=(5, 4))
ax = pam.plot_ZH_regression(rate_analysis_ds, group_by='rate', color_by='rate', cmap='tab10', marker_by='material',
                            flow_stress_key='flow_Stress(MPa)', rate_key='rate', temperature_key='flow_temp(K)', ax=axs,
                            eq_hscale=0.1)
```



[19]:
```
fig, axs = plt.subplots(1, 3, figsize=(3.5, 4))
conformance_matrix(rate_analysis_ds, axs);
```

Tables

[20]:
```
df = pam.make_representative_info(material_analysis_ds, group_by_keys=['material', 'rate', 'temperature'],
                                  group_info_cols=['lnZ_fit_residual', 'lnZ', 'ZH_parameter', 'flow_TC1(C)',
                                                   'flow_Stress(MPa)'])
table = df[['material', 'rate', 'temperature', 'flow_Stress(MPa)', 'flow_TC1(C)', 'ZH_parameter', 'lnZ',
            'quality']].sort_values(['temperature', 'material'])
```

[21]:
```
table_s1 = table[table['rate'] == 1].drop(columns=['rate'])
for col in ['flow_Stress(MPa)', 'flow_TC1(C)', 'lnZ', 'quality']:
    table_s1[col] = table_s1[col].apply(lambda x: f'{x:.3g}')
table_s1.style.to_latex('info/s1_table.tex')
```

temperature	material	flow_Stress(MPa)	flow_TC1(C)	lnZ	quality
360	H560	94	365	31	97.3
400	AC	92.1	405	29.2	95.2
400	H560	76.1	409	29	98.4
400	H580	66.5	447	27.5	94
450	AC	70.8	456	27.2	95.8
450	H560	59.9	458	27.2	99
450	H580	57.2	479	26.4	93.4
500	AC	54.3	506	25.6	94.7
500	H560	48.3	509	25.5	94.1
500	H580	41.3	534	24.7	94

[22]:
```
table_s10 = table[table['rate'] == 10].drop(columns=['rate'])
for col in ['flow_Stress(MPa)', 'flow_TC1(C)',  'lnZ', 'quality']:
    table_s10[col] = table_s10[col].apply(lambda x: f'{x:.3g}')
table_s10.style.to_latex('info/s10_table.tex')
```

temperature	material	flow_Stress(MPa)	flow_TC1(C)	lnZ	quality
300	AC	159	313	35.5	95.5
300	H560	131	314	35.6	99
300	H580	127	324	35	97.8
330	AC	128	359	33.2	99.8
330	H560	119	338	34.3	99.4
330	H580	112	355	33.5	99.4
360	AC	116	372	32.6	94
360	H560	106	368	32.9	99
360	H580	103	369	32.8	97.3
400	AC	89.4	436	29.9	93.4
400	H560	90.7	401	31.5	94.5
400	H580	79	434	30	95.1
450	AC	80.3	469	28.8	95.9
450	H560	76.3	457	29.4	97.8
450	H580	73.3	452	29.5	92.8
500	AC	70.5	500	27.8	95.9
500	H560	61.5	507	27.8	95
500	H580	60.5	500	27.8	93.1

C.4 CS4 Processing Report

Package versions

[1]:
```
import paramaterial as pam
from paramaterial import DataSet, DataItem
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib as mpl
print(pam.__version__)
print(np.__version__)
print(pd.__version__)
print(mpl.__version__)
```

```
0.1.0
1.21.6
1.4.4
3.5.3
```

[2]:
```
pam.check_column_headers('data/01 raw data', exception_headers=['TC4(C)', 'Stroke(mm)', 'wedge(mm)'])
pam.check_for_duplicate_files('data/01 raw data')
```

```
Checking column headers...
First file headers:
        ['Time(sec)', 'Force(kN)', 'Jaw(mm)', 'PowAngle(deg)', 'PTemp',
'Strain', 'Stress(MPa)', 'TC1(C)', 'TC2(C)', 'TC3(C)', 'TC4(C)']
Headers in all files are the same as in the first file, except for ['TC4(C)',
'Stroke(mm)', 'wedge(mm)'].
Checking for duplicate files...
No duplicate files found in "data/01 raw data".
```

Load the data and info into a dataset object (ds is shorthand for dataset).

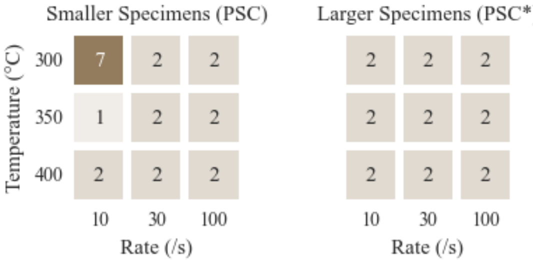
[3]:
```
raw_ds = DataSet('info/01 raw info.xlsx', 'data/01 raw data').sort_by(['test type', 'temperature'])
```

Make the experimental matrix

The tests can be grouped by nominal rate, nominal temperature, and test type. We will make the classic rate-temperature matrices - one for each test type.

[4]:
```
gold_cmap = mpl.colors.LinearSegmentedColormap.from_list("", ["white", (85/255, 49/255, 0)])
mpl.rcParams['axes.facecolor'] = gold_cmap(0.1)
```

[5]:
```
fig, axs = plt.subplots(1, 2, figsize=(4.6,1.6))
heatmap_kwargs = dict(linewidths=4, cbar=False, annot=True, cmap=gold_cmap,  vmin=0, vmax=11)

pam.experimental_matrix(raw_ds.subset({'test type': 'PSC'}).info_table,
                             index='temperature', columns='rate', as_heatmap=True, title='Smaller Specimens (PSC)',
                             xlabel='Rate (/s)', ylabel='Temperature (°C)',
                             tick_params=dict(rotation=0), ax=axs[0], **heatmap_kwargs)

pam.experimental_matrix(raw_ds.subset({'test type': 'PSC*'}).info_table,
                             index='temperature', columns='rate', as_heatmap=True, title='Larger Specimens (PSC*)',
                             xlabel='Rate (/s)', ylabel=' ',
                             tick_params=dict(labelleft=False), ax=axs[1], **heatmap_kwargs);
```



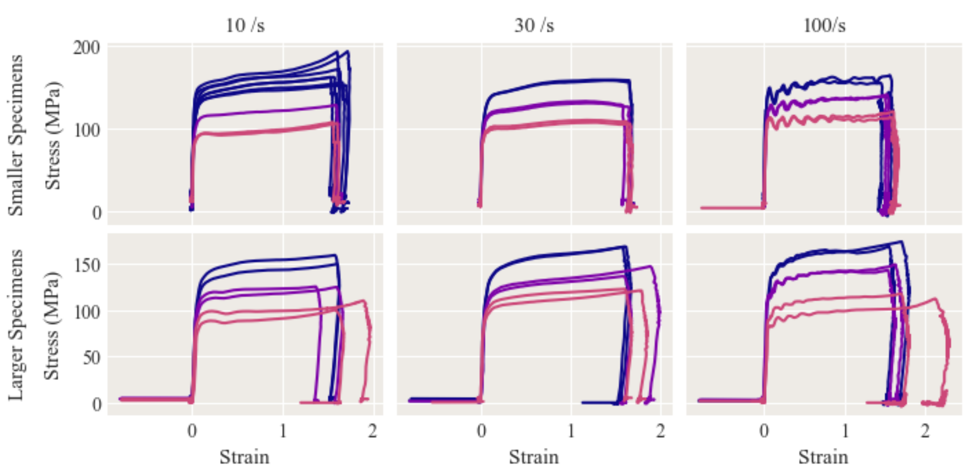
Setup plotting

Now that we know how to group the data, we can set up a plotting function.

[6]:
```
styler = pam.plotting.Styler(color_by='temperature', color_by_label='(°C)', cmap='plasma', color_norm=plt.Normalize(300,500)
                             ).style_to(raw_ds)


def ds_subplot(ds: DataSet, **kwargs):
    return pam.plotting.dataset_subplots(
        ds, shape=(2, 3), figsize=(8, 3.5),
        cols_by='rate', rows_by='test type',
        col_vals=[[10], [30], [100]], row_vals=[['PSC'], ['PSC*']],
        col_titles=['10 /s', '30 /s', '100/s'], row_titles=['Smaller Specimens', 'Larger Specimens'],
        styler=styler, plot_legend=False, subplot_legend=False,
        **kwargs
    )


stress_strain_labels = dict(x='Strain', y='Stress(MPa)', xlabel='Strain', ylabel='Stress (MPa)')
```

Plot the raw data.

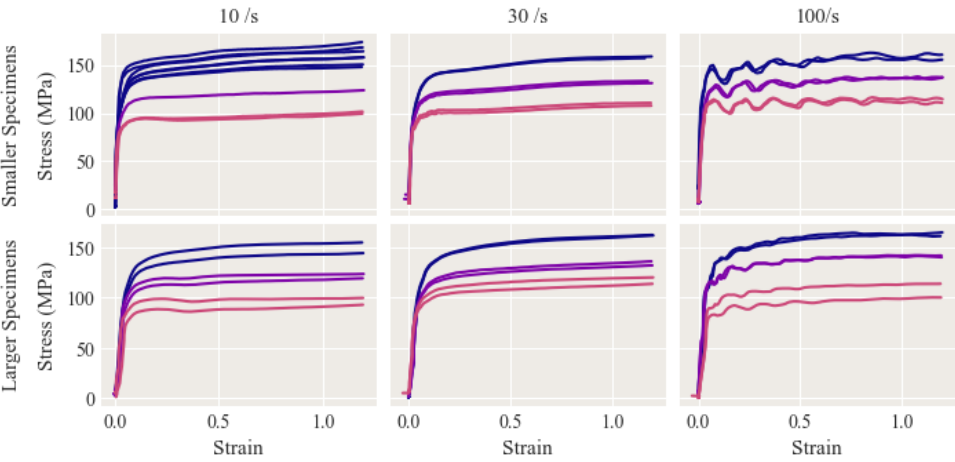
[7]:
```
ds_subplot(raw_ds, **stress_strain_labels);
```



Trim data

[8]:
```
def drop_columns(di: DataItem) -> DataItem:
    for column in ['PowAngle(deg)', 'Power(W)', 'Pram', 'PTemp', 'Stroke(mm)', 'wedge(mm)']:
        if column in di.data.columns:
            di.data = di.data.drop(columns=[column])
    return di


def trim_using_time_step(di: DataItem) -> DataItem:
    t_diff = np.diff(di.data['Time(sec)'])
```

```
    di.data['time diff'] = np.hstack([t_diff[0], t_diff])
    di.data = di.data[di.data['time diff'] < 0.02][1:]
    return di


def remove_trailing_data(di: DataItem):
    di.data = di.data[di.data['Strain'] < 1.2]
    return di

trimmed_ds = raw_ds.apply(drop_columns).apply(trim_using_time_step).apply(remove_trailing_data)
ds_subplot(trimmed_ds, **stress_strain_labels);
```



Find flow stress values

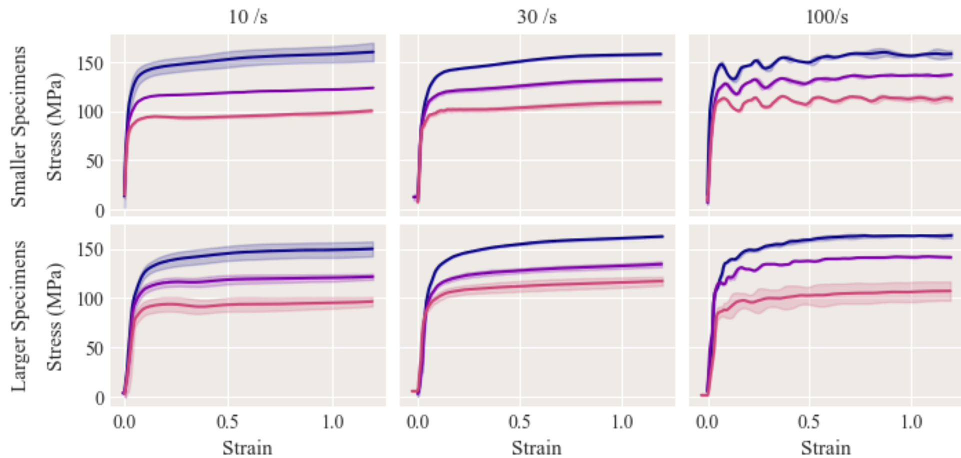
Determine the stress, temperature for each test. Note that while nominal temperature is given, the actual test conditions are slightly different. The actual strain-rate does not differ significantly from the nominal rate for these tests.

[9]:
```
rates_ds = pam.calculate_strain_rate(trimmed_ds, time_key='Time(sec)')

processed_ds = pam.find_flow_stress_values(rates_ds, flow_strain=(0.8, 0.9),
                              stress_key='Stress(MPa)', temperature_key='TC1(C)', rate_key='Strain_Rate'
                              ).sort_by('rate')
```

[10]:
```
processed_ds.write_output('info/02 processed info.xlsx', 'data/02 processed data')
```

Make representative curves

[11]:
```
pam.make_representative_data(processed_ds, 'info/03 repr trim info.xlsx', 'data/03 repr trim data',
                             repres_col='Stress(MPa)', group_by_keys=['temperature', 'test type', 'rate'],
                             interp_by='Strain', group_info_cols=['flow_Stress(MPa)'])

repr_ds = DataSet('info/03 repr trim info.xlsx', 'data/03 repr trim data', test_id_key='repres_id').sort_by('temperature')

ds_subplot(repr_ds, x='Strain', y='Stress(MPa)', xlabel='Strain', ylabel='Stress (MPa)',
               fill_between=('down_std_Stress(MPa)', 'up_std_Stress(MPa)'));
```

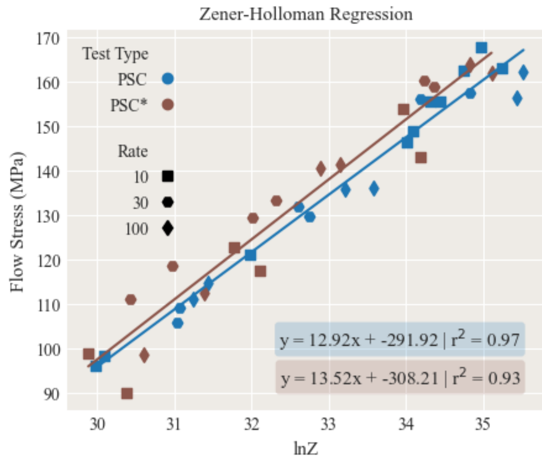


Zener-Holloman Analysis

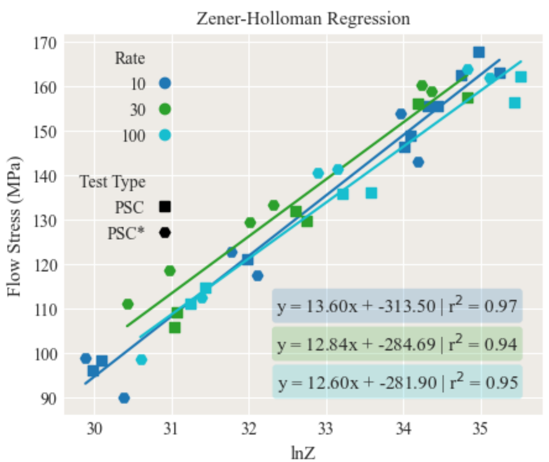
[12]:
```
default_cmap = plt.cm.get_cmap('RdYlGn')
colors = default_cmap(np.arange(default_cmap.N))
colors[0] = (1, 1, 1, 1)
custom_cmap = mpl.colors.ListedColormap(colors)

def conformance_matrix(_ds: DataSet):
    fig, axs = plt.subplots(1, 2, figsize=(4.6, 1.6))
    heatmap_kwargs = dict(linewidths=4, cbar=False, annot=True, fmt='.1f', cmap=custom_cmap, vmin=85, vmax=100)
    pam.make_quality_matrix(_ds.subset({'test type': 'PSC'}).info_table, index='temperature', columns='rate',
                            flow_stress_key='flow_Stress(MPa)', as_heatmap=True, title='Smaller Specimens (PSC)',
                            xlabel='Rate (/s)', ylabel='Temperature (°C)', tick_params=dict(rotation=0), ax=axs[0],
                            **heatmap_kwargs)
    pam.make_quality_matrix(_ds.subset({'test type': 'PSC*'}).info_table, index='temperature', columns='rate',
                            flow_stress_key='flow_Stress(MPa)', as_heatmap=True, title='Larger Specimens (PSC*)',
                            xlabel='Rate (/s)', ylabel=' ', tick_params=dict(labelleft=False), ax=axs[1],
                            **heatmap_kwargs)
```

[13]:
```
processed_ds.info_table['Q_activation'] = 155000.
processed_ds.info_table['flow_temp(K)'] = processed_ds.info_table['flow_TC1(C)'] + 273.15
analysis_ds = processed_ds.apply(pam.calculate_ZH_parameter, rate_key='flow_Strain_Rate', temperature_key='flow_temp(K)')
```

[14]:
```
type_analysis_ds = pam.apply_ZH_regression(analysis_ds.copy(), flow_stress_key='flow_Stress(MPa)', group_by='test type')
ax = pam.plot_ZH_regression(type_analysis_ds, group_by='test type', color_by='test type', marker_by='rate', cmap='tab20',
                            flow_stress_key='flow_Stress(MPa)', rate_key='flow_Strain_Rate', temperature_key='flow_temp(K)',figsize=(5,4
conformance_matrix(type_analysis_ds)
```



[15]:
```
rate_analysis_ds = pam.apply_ZH_regression(analysis_ds.copy(), flow_stress_key='flow_Stress(MPa)', group_by='rate')
ax = pam.plot_ZH_regression(rate_analysis_ds, group_by='rate',
                       color_by='rate', marker_by='test type', cmap='tab10',
                       flow_stress_key='flow_Stress(MPa)', rate_key='flow_Strain_Rate', temperature_key='flow_temp(K)',figsize=(5,4
conformance_matrix(rate_analysis_ds)
```



Tables

[16]:
```
df = pam.make_representative_info(type_analysis_ds, group_by_keys=['test type', 'rate', 'temperature'],
                                  group_info_cols=['lnZ_fit_residual', 'lnZ', 'ZH_parameter', 'flow_TC1(C)',
                                                   'flow_Stress(MPa)'])
table = df[['test type', 'rate', 'temperature', 'flow_Stress(MPa)', 'flow_TC1(C)', 'ZH_parameter', 'lnZ',
            'quality']].sort_values(['temperature', 'test type'])
table = table.reset_index().drop(columns=['index'])
table = table[['temperature', 'test type', 'rate', 'flow_Stress(MPa)', 'flow_TC1(C)',  'lnZ', 'quality']]
table_s10 = table[table['rate'] == 10].drop(columns=['rate'])
table_s30 = table[table['rate'] == 30].drop(columns=['rate'])
table_s100 = table[table['rate'] == 100].drop(columns=['rate'])
```

[17]:
```
for col in ['flow_Stress(MPa)', 'flow_TC1(C)', 'lnZ', 'quality']:
    table_s10[col] = table_s10[col].apply(lambda x: f'{x:.3g}')

table_s10.style.to_latex('info/s10_table.tex')
```

temperature	test type	flow_Stress(MPa)	flow_TC1(C)	lnZ	quality
300	PSC	157	320	34.8	98.4
300	PSC*	148	328	34.3	97.2
350	PSC	121	371	32.2	99.7
350	PSC*	120	370	32.1	97
400	PSC	97.1	417	30.2	99.1
400	PSC*	94.5	415	30.3	94.9

[18]:

```
for col in ['flow_Stress(MPa)', 'flow_TC1(C)',  'lnZ', 'quality']:
    table_s30[col] = table_s30[col].apply(lambda x: f'{x:.3g}')

table_s30.style.to_latex('info/s30_table.tex')
```

temperature	test type	flow_Stress(MPa)	flow_TC1(C)	lnZ	quality
300	PSC	157	349	34.7	98.3
300	PSC*	159	346	34.5	97.6
350	PSC	131	389	32.9	99.7
350	PSC*	131	396	32.4	96.4
400	PSC	108	428	31.2	98.4
400	PSC*	115	435	30.9	93

[19]:

```
for col in ['flow_Stress(MPa)', 'flow_TC1(C)', 'lnZ', 'quality']:
    table_s100[col] = table_s100[col].apply(lambda x: f'{x:.3g}')

table_s100.to_latex('info/s100_table.tex')
```

temperature	test type	flow_Stress(MPa)	flow_TC1(C)	lnZ	quality
300	PSC	159	351	35.7	95.6
300	PSC*	163	350	35.2	99.1
350	PSC	136	387	33.6	97.4
350	PSC*	141	398	33.2	98
400	PSC	113	435	31.5	99.8
400	PSC*	106	452	31.2	95