

BME Faculty of Mechanical Engineering	EXPERIMENTAL METHODS	Name: Kelle Gergő
Dep. of Applied Mechanics	1st HOMEWORK	Neptun code: GBBNUL
2024/25 I.	Deadline: see Moodle	Late submission <input type="checkbox"/> Correction <input type="checkbox"/>
Declaration: With my signature I declare, that I did my homework myself, everything written in there is my knowledge.	Signature:	

We only correct homeworks that correspond with formal requirements. Correction or late submission is only possible until the late submission deadline.

### Problem

An open-cell polyurethane foam material has been tested using uniaxial relaxation and cyclic compression tests with an Instron 3345 Single Column Universal Testing System, while the load was measured by an Instron model 2519-107 5kN load cell. The displacement was measured directly from the cross-head position. The measurement layout is depicted in Figure 1, while the exact dimensions of the specimens are listed in the tables below. The initial tool separation was  $H_0$ . The deformation of the specimen is assumed to be homogeneous.

The link for the measurement data can be found at the bottom of the 2nd page below the table of test parameters!

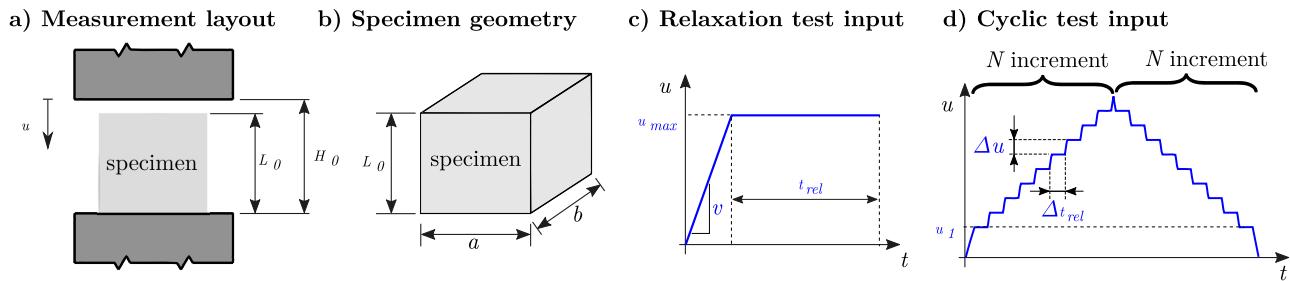


Figure 1: Measurement layout and displacement inputs

### Task A

The relaxation test comprises of two phases (see Fig. 1/c): an unloading phase up to the maximal displacement  $u_{max}$  with cross-head speed  $v$ , and a relaxation phase for time  $t_{rel}$ . The measurement was also recorded with digital camera. The raw data of the measurement contains the measured time [s], extension [mm] and load [N] values and the video of the measurement in .mp4 format.

- Determine the exact height of the relaxation test specimen ( $l_0^{rel}$ ); obtain and plot the compensated force-displacement ( $F - u$ ) and force-time ( $F - t$ ) curves using slack correction method!
- Determine and plot the measured engineering stress - engineering strain ( $P - \varepsilon$ ), engineering stress – stretch ( $P - \lambda$ ) and engineering stress-time ( $P - t$ ) curves!
- Determine and plot the cross-directional stretch characteristics ( $\lambda - \lambda_T$ ) using a self-developed image processing method on the video recording with a suitable programming language (MatLab, Wolfram, Python, etc...)!
- Determine the maximal  $\lambda_{T,max}$  value!
- Plot the true stress – true strain curve ( $\sigma - \varepsilon^{true}$ )!

## Task B

After that, a cyclic compression test was performed on another specimen from the same material. The cyclic tests consist of  $N$  increments both in the uploading and unloading phases (see Fig. 1/d). The displacement of the first step in the uploading (and the last step of the unloading) was  $u_1$ , while in all other increments the displacement was  $\Delta u$ . The cross-head speed was  $v_{cyc}$ . After each increment (except the  $n^{th}$ ), a relaxation phase for time  $\Delta t_{rel}$  was inserted. The raw data of the measurement contains the measured time [s], extension [mm] and load [N] values.

1. Determine the exact height of the cyclic test specimen ( $L_0^{cyc}$ ); obtain and plot the compensated force-displacement ( $F - u$ ) and force-time ( $F - t$ ) curves using slack correction method!
2. Plot the measured engineering stress - engineering strain ( $P - \varepsilon$ ), engineering stress – stretch ( $P - \lambda$ ) and engineering stress-time ( $P - t$ ) curves!
3. List and plot the data points corresponding to the long-term, pure elastic engineering-stress – stretch ( $P - \lambda$ ) curve, by interpolating between endpoints of the relaxation phases in the uploading and unloading curves.
4. Illustrate on the same graph the engineering stress-stretch curves ( $P - \lambda$ ) obtained from the relaxation test and pure elastic approximation. Determine the average of  $\eta = P^{rel}(\lambda)/P^{long}(\lambda)$  ratio of the corresponding long-term and relaxation engineering stress values!

### Formal requirements

Create a PDF report on the measurement results and the post-processing using document processor software (e.g. Word, LaTeX). The first page must be the signed cover sheet. Please also attach the commented program code of the image processing. The header and the numerical results have to be filled. Short comments and explanations must be given to each step, to such an extent that the work may be reproduced even by colleagues who are not specialized in the given topic. All questions must be answered! Do not upload pure program codes, it won't be accepted.

### Data

Parameters of the relaxation test:

Specimen number	$H_0$ [mm]	$u_{max}$ [mm]	$a$ [mm]	$b$ [mm]	$v$ [mm/min]	$t_{rel}$ [s]
6	75	56	76.24	74.91	120	30

Download relaxation test data: <https://www.mm.bme.hu/edu/msc-en/expmeth/0hw1/relax6.zip>

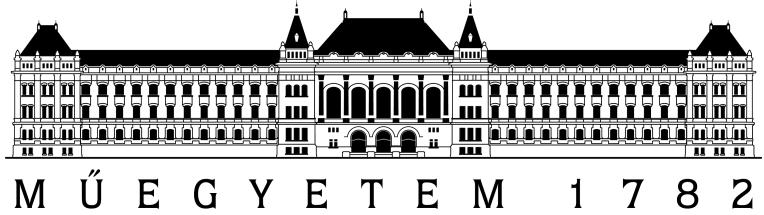
Parameters of the cyclic test:

Specimen number	$H_0$ [mm]	$u_1$ [mm]	$\Delta u$ [mm]	$\Delta t_{rel}$ [s]	$N$ [-]	$v_{cyc}$ [mm/min]	$a$ [mm]	$b$ [mm]
18	75	10.5	5.5	30	9	40	77.71	75.11

Download cyclic test raw data: <https://www.mm.bme.hu/edu/msc-en/expmeth/0hw1/cyclic18.zip>

### Results

$L_0^{rel}$ [mm]	$\lambda_{T,max}$ [-]	$L_0^{cyc}$ [mm]	$\eta_{avg}$ [-]
69.17	1.02376	68.5	1.425



BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS  
FACULTY OF MECHANICAL ENGINEERING

---

# Experimental methods in solid mechanics

## I. Homework

---

Kelle Gergő

November 2, 2024

# Contents

<b>1 Task A - Relaxation</b>	<b>5</b>
1.1 Task A1 . . . . .	5
1.2 Task A2 . . . . .	6
1.3 Task A3 . . . . .	7
1.4 Task A4 . . . . .	8
1.5 Task A5 . . . . .	8
<b>2 Task B - Cyclic Compression</b>	<b>9</b>
2.1 Task B1 . . . . .	9
2.2 Task B2 . . . . .	10
2.3 Task B3 . . . . .	10
2.4 Task B4 . . . . .	11
<b>A Code Appendix</b>	<b>13</b>

# 1 Task A - Relaxation

## 1.1 Task A1

**Task description:** Determine the exact height of the relaxation test specimen ( $L_0^{rel}$ ); obtain and plot the compensated force - displacement ( $F$  -  $u$ ) and force-time ( $F$  -  $t$ ) curves using slack correction method!

The initial height of the relaxation specimen,  $L_0^{rel}$ , can be precisely calculated from the measured data. When the load begins to increase, it indicates initial contact between the foam and the compression head. Thus, the specimen height is determined as the initial tool separation minus the extension at this point:

$$L_0^{rel} = H_0 - u_0 \approx 69.17 \text{ mm} \quad (1)$$

To conduct slack correction, first the inflection point has to be located on the load-displacement curve and a tangent line at that point has to be drawn. Next, I've found the intersection of this tangent with the x-axis and eliminated the measurement data before this point. After that I've substituted the region with a linear segment. This method redefined the zero-displacement point as the following figures shows.

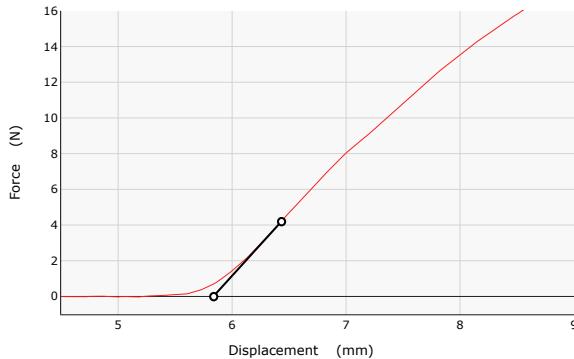


Figure 1: *Slack correction - tangential linear part*

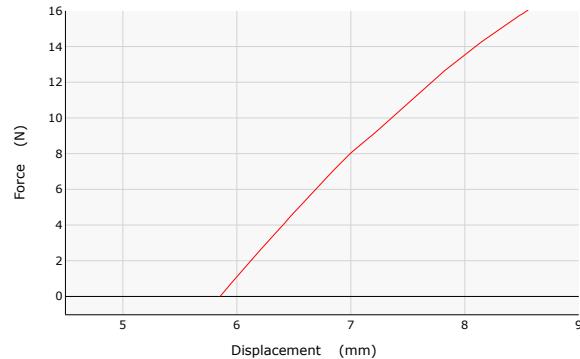


Figure 2: *Slack corrected part*

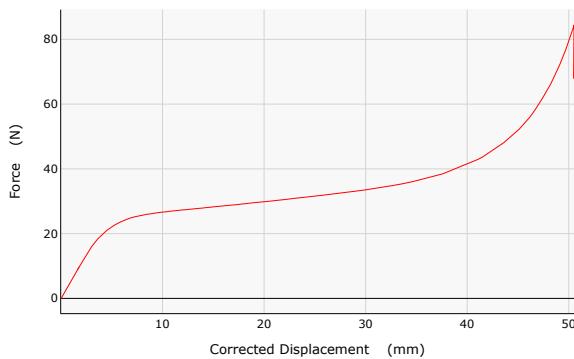


Figure 3: *Force - compensated displacement plot*

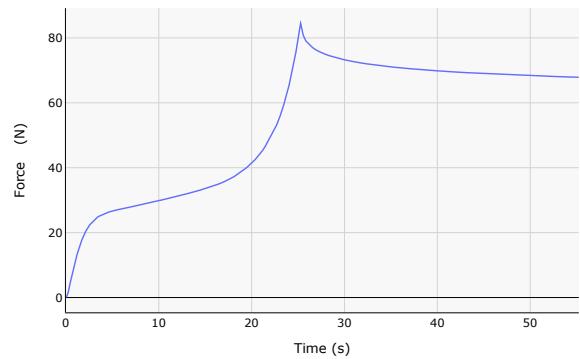


Figure 4: *Force - time plot*

## 1.2 Task A2

**Task description:** Determine and plot the measured engineering stress- engineering strain ( $P - \varepsilon$ ), engineering stress - stretch ( $P - \lambda$ ) and engineering stress-time ( $P - t$ ) curves!

First let's clarify the used formulas. The cross-sectional area can be determined as

$$A = a \cdot b = 5711.1384 \text{ [mm}^2\text{].} \quad (2)$$

The engineering stress can be calculated as

$$P = \frac{F}{A}. \quad (3)$$

The stretch can be calculated as

$$\lambda = 1 - \frac{u}{L_0^{rel}}. \quad (4)$$

The engineering strain can be calculated as:

$$\varepsilon = \lambda - 1. \quad (5)$$

Using these formulas I've created the necessary plots.

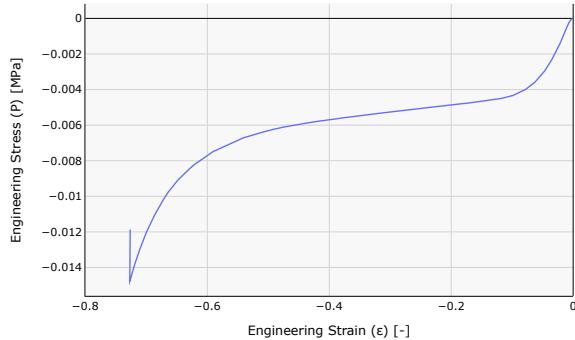


Figure 5: Engineering Stress - Engineering Strain Curve

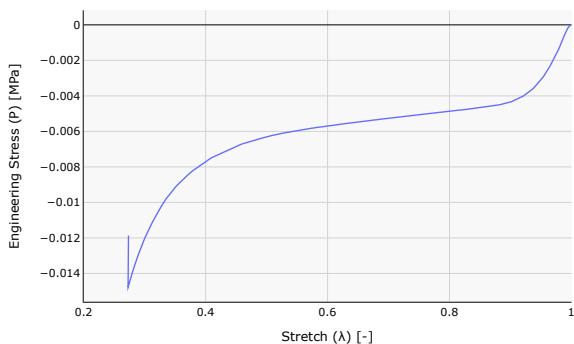


Figure 6: Engineering Stress - Stretch Curve

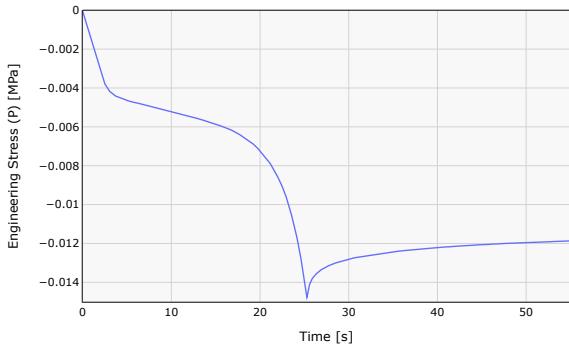


Figure 7: Engineering Stress - Time Curve

### 1.3 Task A3

**Task description:** Determine and plot the cross-directional stretch characteristics ( $\lambda - \lambda_T$ ) using a self-developed image processing method on the video recording with a suitable programming language (MatLab, Wolfram, Python, etc...)!

The implementation of the image processing method was conducted using Python. The process began with video cropping to focus on the relevant area of interest. Next, each frame was converted to grayscale to simplify the data and enhance contrast. A thresholding technique was then applied to produce a binary image, transforming the grayscale frames into black and white representations. To quantify the cross-directional stretch, the sum of white pixels was calculated along the video duration at half-second intervals.

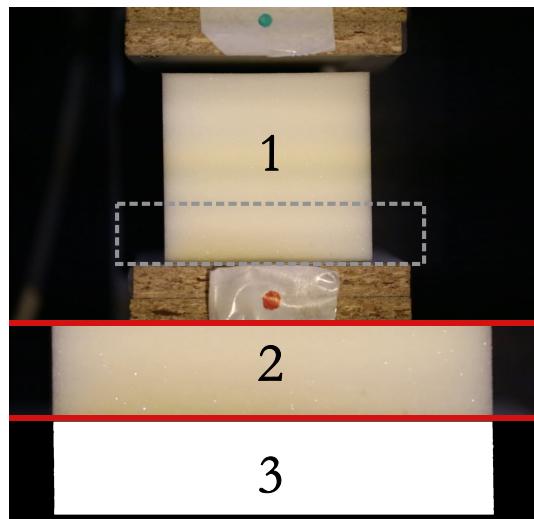


Figure 8: Video processing

After processing the video cross-directional stretch characteristics ( $\lambda - \lambda_T$ ) plot can be done after evening out the two data set row numbers.

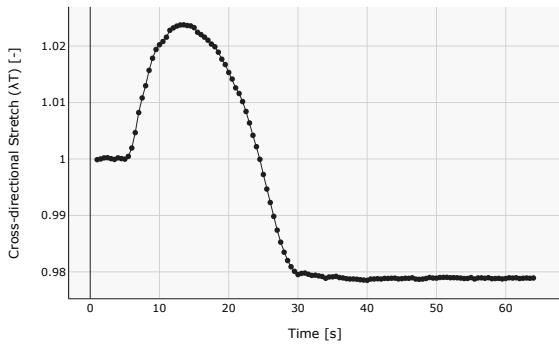


Figure 9: *Cross-dir. Stretch Characteristics*

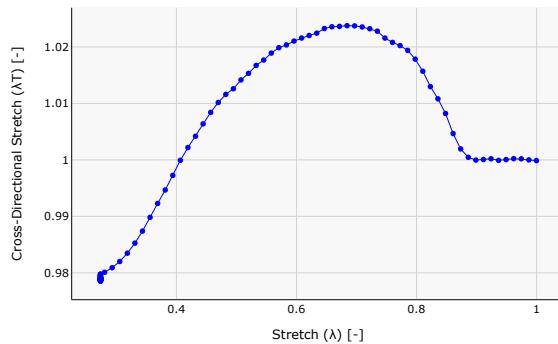


Figure 10: *Cross-dir. Stretch - Stretch plot*

The plots displayed anticipated outcomes, indicating that while foams are easily compressed, they exhibit minimal transverse stretch in the material.

## 1.4 Task A4

**Task description:** Determine the maximal  $\lambda_T,\max$  value!

$$\max(\lambda_T) = 1.02376 \quad (6)$$

## 1.5 Task A5

**Task description:** Plot the true stress - true strain curve ( $\sigma - \varepsilon^{\text{true}}$ )!

True stress we need the actual cross-section area that can be calculated as

$$\sigma_{\text{true}} = \frac{F}{\lambda_T^2 a b} \quad (7)$$

The true strain can be calculated with the following equation

$$\varepsilon_{\text{true}} = \ln(\lambda) \quad (8)$$

After the calculation the number of values in each dataset should be matched in order to plot properly, see the results on the Figure 11 below.

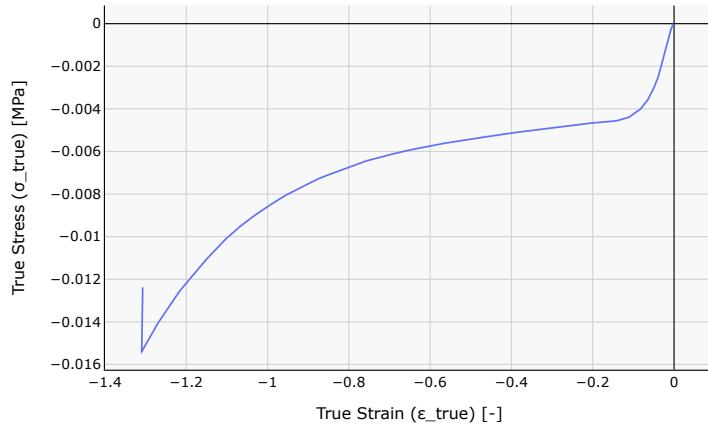


Figure 11: *True stress - True strain plot*

A true strain of  $-1.3$  implies that the material's current length is only about  $e^{-1.3} \approx 0.27$  of its original length. This suggests about a 73% reduction in length, indicating substantial compression.

## 2 Task B - Cyclic Compression

### 2.1 Task B1

**Task description:** Determine the exact height of the cyclic test specimen ( $L_0^{\text{cyc}}$ ); obtain and plot the compensated force - displacement ( $F$  -  $u$ ) and force - time ( $F$  -  $t$ ) curves using slack correction method!

The initial height of the cyclic test specimen,  $L_0^{\text{cyc}}$ , can be calculated in the same manner as for the relaxation test. Furthermore, the required plots have been generated analogously to those for the relaxation test.

$$L_0^{\text{cyc}} = H_0 - u_{0,\text{cyc}} = 68.5 \text{ mm} \quad (9)$$

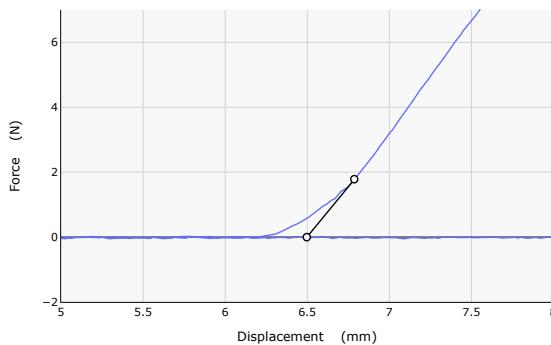


Figure 12: *Slack corr. - tangential linear part*

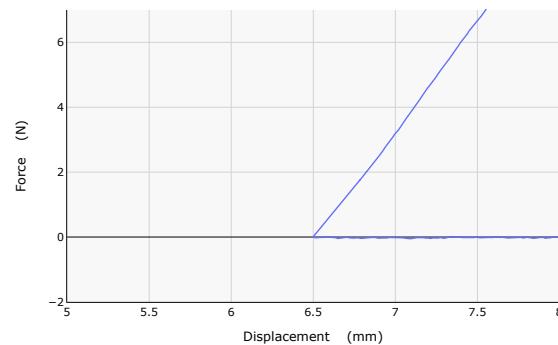


Figure 13: *Slack corrected part*

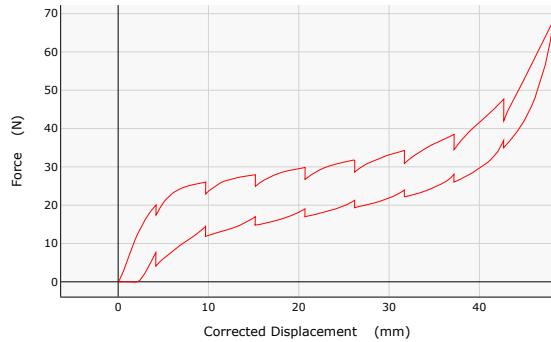


Figure 14: *Force - compensated disp. plot*

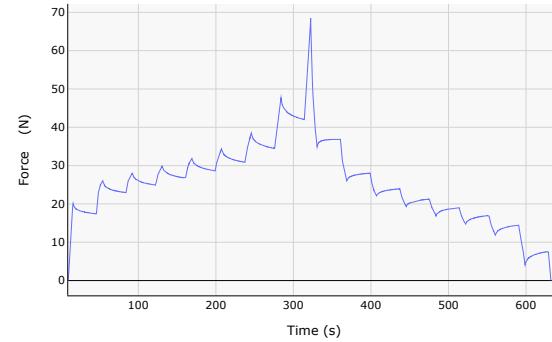


Figure 15: *Force - time plot*

## 2.2 Task B2

**Task description:** Plot the measured engineering stress- engineering strain ( $P - \varepsilon$ ), engineering stress - stretch ( $P - \lambda$ ) and engineering stress-time ( $P - t$ ) curves!

To plot the required diagrams; engineering stress, engineering strain, and stretch must be calculated. This calculation is performed using the same equations established for the relaxation test; however, in this case, I utilize the results from the cyclic test.

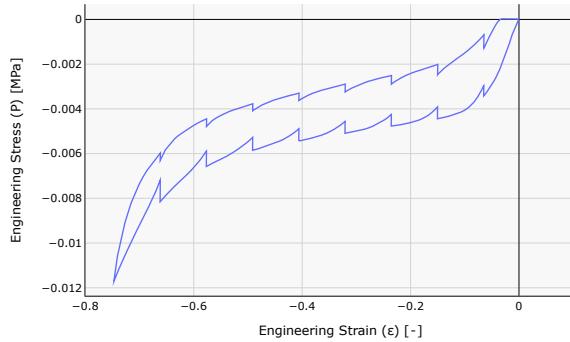


Figure 16: Engineering Stress - Engineering Strain Curve

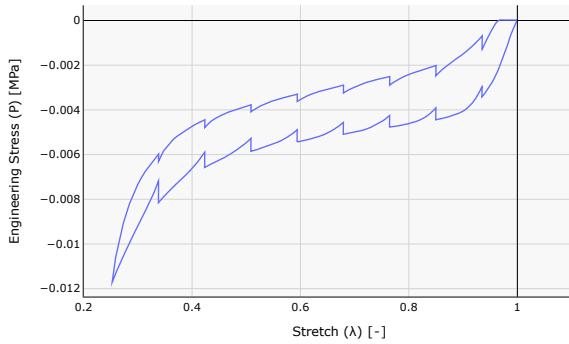


Figure 17: Engineering Stress - Stretch Curve

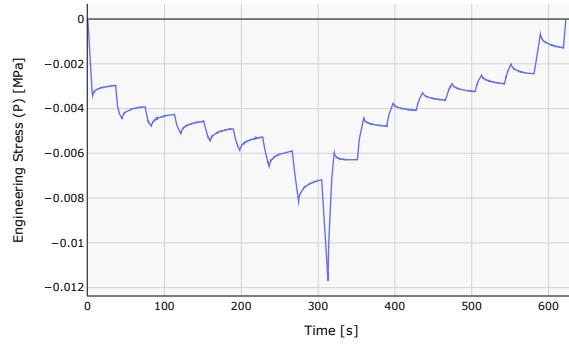


Figure 18: Engineering Stress - Time Curve

## 2.3 Task B3

**Task description:** List and plot the data points corresponding to the long-term, pure elastic engineering stress - stretch ( $P - \lambda$ ) curve, by interpolating between endpoints of the relaxation phases in the uploading and unloading curves.

In order to generate the long-term, pure elastic engineering stress-stretch ( $P - \lambda$ ) curve, I began by identifying the local minima and maxima of the uploading and unloading curves. After that, I've calculated the midpoints by averaging the corresponding values of these local extrema. Finally, I fitted a line through these midpoints to create the desired interpolation, resulting in the pure elastic approximation curve shown in Figure 19.

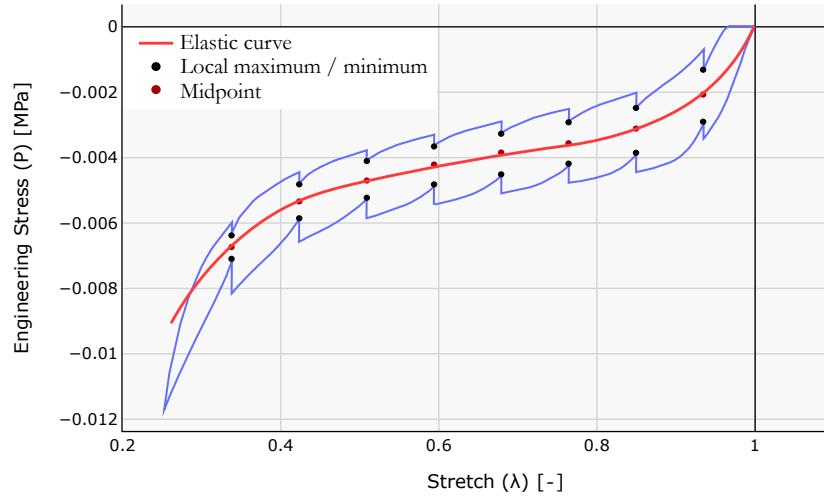


Figure 19: Engineering Stress - Stretch with fitted pure elastic approximation curve

## 2.4 Task B4

**Task description:** Illustrate on the same graph the engineering stress-stretch curves ( $P - \lambda$ ) obtained from the relaxation test and pure elastic approximation. Determine the average of  $\eta = P^{rel}(\lambda)/P^{long}(\lambda)$  ratio of the corresponding long-term and relaxation engineering stress values!!

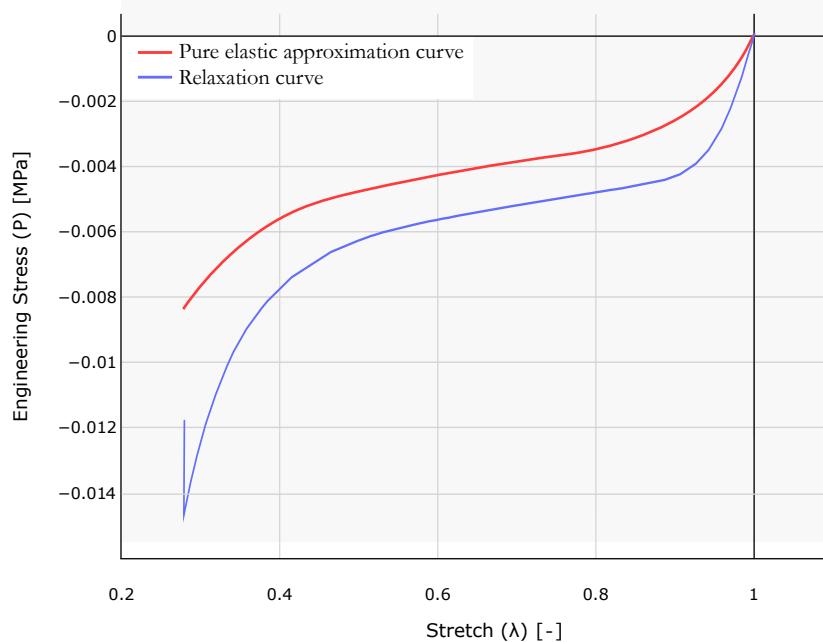


Figure 20: Pure Elastic approximation and relaxation test result

As for the ratio of the corresponding long-term and relaxation engineering stress values, I chose a straightforward approach using image processing. I converted Figure 20 into two separate binary

images, summed up the white pixels, and calculated the ratio as:

$$\eta = \frac{P^{\text{rel}}(\lambda)}{P^{\text{long}}(\lambda)} = 1.425 \quad (10)$$

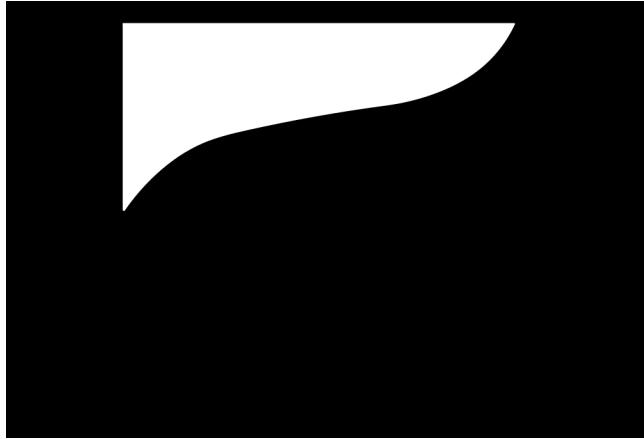


Figure 21: *Binary picture of the pure elastic approximation*

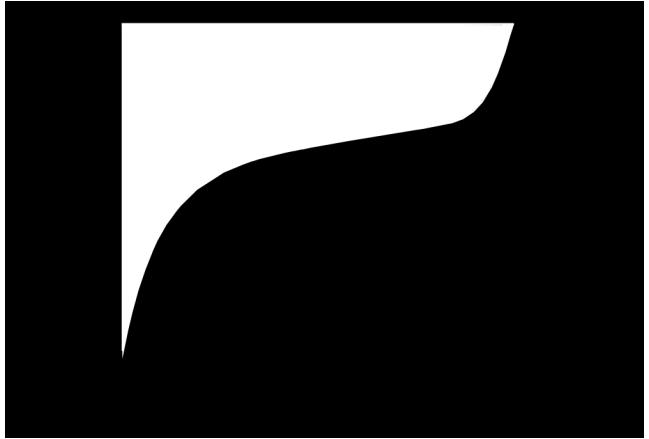


Figure 22: *Binary picture of the relaxation test result*

## A Code Appendix

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[21]:
5
6
7 clear all
8
9
10 # In[22]:
11
12
13 def style_plot(fig, x_title, y_title, title):
14     # Add gradient background using shapes
15     fig.update_layout(
16         title=title,
17         plot_bgcolor='rgba(255, 255, 255, 0)',    # Transparent plot background
18         shapes=[
19             dict(
20                 type='rect',
21                 xref='paper', yref='paper',
22                 x0=0, x1=1, y0=0, y1=1,
23                 fillcolor='rgba(211, 211, 211, 0.15)',   # Grey with 15% opacity
24                 line=dict(width=0)
25             ),
26         ],
27         xaxis=dict(
28             title=x_title,
29             gridcolor='lightgrey',    # Lighter grid color for more hollow effect
30             gridwidth=1,
31             zerolinecolor='black',   # Zero line color
32             zerolinewidth=2,
33             showgrid=True,
34             linecolor='black' # X-axis line color
35         ),
36         yaxis=dict(
37             title=y_title,
38             gridcolor='lightgrey',    # Lighter grid color for more hollow effect
39             gridwidth=1,
40             zerolinecolor='black',   # Zero line color
41             zerolinewidth=2,
42             showgrid=True,
43             linecolor='black' # Y-axis line color
44         ),
45         font=dict(color='black'),
46     )
47
48     # Set colors for traces
49     if len(fig.data) >= 1:
50         fig.update_traces(line=dict(color='red'), selector=dict(name='
51             Compensated F-u'))) # Default color for the first trace
52         if len(fig.data) >= 2:

```

```
52     fig.data[1].line.color = 'green' # Set the second trace color to green
53
54
55 # In[23]:
56
57
58 ## TASK A - Preparation
59
60
61 # In[24]:
62
63
64 # Import necessary libraries
65 import pandas as pd
66 import plotly.graph_objs as go
67
68 # Define basic parameters
69 H_0 = 75 # mm, initial tool separation
70 u_max = 56 # mm, maximal displacement during unloading
71 a = 76.24 # mm, specimen width
72 b = 74.91 # mm, specimen length
73 v = 120 # mm/min, cross-head speed during unloading
74 t_rel = 30 # s, relaxation time
75
76 # Load the CSV file into a DataFrame
77 file_name = 'Relax_RawData_6.csv'
78 data = pd.read_csv(file_name)
79
80 # Convert columns to appropriate data types
81 data['Time'] = pd.to_numeric(data.iloc[:, 0], errors='coerce') # Convert Time
     to float
82 data['Extension'] = pd.to_numeric(data.iloc[:, 1], errors='coerce') # Convert
     Extension to float
83 data['Load'] = pd.to_numeric(data.iloc[:, 2], errors='coerce') # Convert Load
     to float
84
85 # Extract the columns: Time, Extension, Load
86 time = data['Time']
87 extension = data['Extension']
88 load = data['Load']
89
90 # Plot Extension vs Time
91 ext_vs_time = go.Scatter(x=time, y=extension, mode='lines', name='Extension vs
     Time')
92 layout_ext = go.Layout(
93     title='Extension vs Time',
94     xaxis=dict(title='Time (s)', tickformat='.1f'), # 1 decimal for Time
95     yaxis=dict(title='Extension (mm)', tickformat='.2f'), # 2 decimals for
     Extension
96 )
97 fig_ext = go.Figure(data=[ext_vs_time], layout=layout_ext)
98
99 # Plot Load vs Time
100 load_vs_time = go.Scatter(x=time, y=load, mode='lines', name='Load vs Time')
```

```
101 layout_load = go.Layout(
102     title='Load vs Time',
103     xaxis=dict(title='Time (s)', tickformat='.1f'), # 1 decimal for Time
104     yaxis=dict(title='Load (N)', tickformat='.2f'), # 2 decimals for Load
105 )
106 fig_load = go.Figure(data=[load_vs_time], layout=layout_load)
107
108 # Show both plots
109 fig_ext.show()
110 fig_load.show()
111
112
113 # In [25]:
114
115
116 # Import necessary libraries
117 import pandas as pd
118 import plotly.graph_objs as go
119
120 # Define basic parameters
121 H_0 = 75 # mm, initial tool separation
122 u_max = 56 # mm, maximal displacement during unloading
123 a = 76.24 # mm, specimen width
124 b = 74.91 # mm, specimen length
125 v = 120 # mm/min, cross-head speed during unloading
126 t_rel = 30 # s, relaxation time
127
128 # Load the CSV file into a DataFrame
129 file_name = 'Relax_RawData_6.csv'
130 data = pd.read_csv(file_name)
131
132 # Convert columns to appropriate data types
133 data['Time'] = pd.to_numeric(data.iloc[:, 0], errors='coerce') # Convert Time
134     to float
135 data['Extension'] = pd.to_numeric(data.iloc[:, 1], errors='coerce') # Convert
136     Extension to float
137 data['Load'] = pd.to_numeric(data.iloc[:, 2], errors='coerce') # Convert Load
138     to float
139
140 # Extract the columns: Time, Extension, Load
141 time = data['Time']
142 extension = data['Extension']
143 load = data['Load']
144
145 # Create the plot for Extension vs Time
146 ext_vs_time = go.Scatter(x=time, y=extension, mode='lines', name='Extension vs
147     Time')
148 layout_ext = go.Layout(
149     title='Extension vs Time',
150     xaxis=dict(title='Time (s)', tickformat='.1f'), # 1 decimal for Time
151     yaxis=dict(title='Extension (mm)', tickformat='.2f'), # 2 decimals for
152     Extension
153 )
154 fig_ext = go.Figure(data=[ext_vs_time], layout=layout_ext)
```

```

150
151 # Create the plot for Load vs Time
152 load_vs_time = go.Scatter(x=time, y=load, mode='lines', name='Load vs Time')
153 layout_load = go.Layout(
154     title='Load vs Time',
155     xaxis=dict(title='Time (s)', tickformat='.1f'), # 1 decimal for Time
156     yaxis=dict(title='Load (N)', tickformat='.2f'), # 2 decimals for Load
157 )
158 fig_load = go.Figure(data=[load_vs_time], layout=layout_load)
159
160 # Apply the styling to both figures
161 style_plot(fig_ext, 'Time (s)', 'Extension (mm)', 'Extension vs Time')
162 style_plot(fig_load, 'Time (s)', 'Load (N)', 'Load vs Time')
163
164 # Show both plots
165 fig_ext.show()
166 fig_load.show()
167
168
169 # TASK A - TASK 1
170
171 # In[157]:
172
173
174 # Identifying the slack region (first non-zero significant load)
175 slack_threshold = 0.1 # N, threshold for detecting when the load becomes
176 # significant
177 slack_index = (load > slack_threshold).idxmax() # Get index of first
178 # significant load
179
180 # Slice the data from slack_index onwards
181 time_corrected = time[slack_index:].reset_index(drop=True) # Time starting
182 # from slack point
183 u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(
184     drop=True) # Corrected extension
185 load_corrected = load[slack_index:].reset_index(drop=True) # Corresponding
186 # load values
187
188 # Plot compensated Force-Displacement (F-u) curve
189 compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
190     mode='lines', name='Compensated F-u')
191 layout_fu = go.Layout(
192     title='Compensated Force-Displacement (F-u) Curve',
193     xaxis=dict(title='Corrected Displacement u (mm)'),
194     yaxis=dict(title='Force F (N)'), )
195 fig_fu = go.Figure(data=[compensated_force_displacement], layout=layout_fu)
196
197 # Plot Force-Time (F-t) curve using corrected time
198 force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name=
199     'F-t')
200 layout_ft = go.Layout(
201     title='Force-Time (F-t) Curve',
202     xaxis=dict(title='Time (s)'), )

```

```

197     yaxis=dict(title='Force F (N)'),
198 )
199 fig_ft = go.Figure(data=[force_time], layout=layout_ft)
200
201 # Show both plots
202 fig_fu.show()
203 fig_ft.show()
204
205
206 # In[158]:
207
208
209 import pandas as pd
210 import plotly.graph_objs as go
211
212 # Load the data
213 file_name = 'Relax_RawData_6.csv'
214 data = pd.read_csv(file_name)
215
216 # Ensure numeric conversion for relevant columns
217 data['Time'] = pd.to_numeric(data['Time'], errors='coerce')
218 data['Extension'] = pd.to_numeric(data['Extension'], errors='coerce')
219 data['Load'] = pd.to_numeric(data['Load'], errors='coerce')
220
221 # Drop rows with NaN values in essential columns
222 data = data.dropna(subset=['Time', 'Extension', 'Load'])
223
224 # Extract the columns
225 time = data['Time']
226 extension = data['Extension']
227 load = data['Load']
228
229 # Identifying the slack region (first non-zero significant load)
230 slack_threshold = 0.1 # N, threshold for detecting significant load
231 slack_index = (load > slack_threshold).idxmax() # Get index of first
    significant load
232
233 # Slice the data from slack_index onwards
234 time_corrected = time[slack_index:].reset_index(drop=True) # Time starting
    from slack point
235 u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(
    drop=True) # Corrected extension
236 load_corrected = load[slack_index:].reset_index(drop=True) # Corresponding
    load values
237
238 # Plot compensated Force-Displacement (F-u) curve
239 compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
    mode='lines', name='Compensated F-u')
240 fig_fu = go.Figure(data=[compensated_force_displacement])
241
242 # Apply styling to the F-u figure
243 style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated
    Force-Displacement (F-u) Curve')
244

```

```

245 # Plot Force-Time (F-t) curve using corrected time
246 force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name=
247   'F-t')
248 fig_ft = go.Figure(data=[force_time])
249
250 # Apply styling to the F-t figure
251 style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
252
253 # Show both plots
254 fig_fu.show()
255 fig_ft.show()
256
257 # In[166]:
258
259
260 import pandas as pd
261 import plotly.graph_objs as go
262
263 # Load the data
264 file_name = 'Relax_RawData_6.csv'
265 data = pd.read_csv(file_name)
266
267 # Ensure numeric conversion for relevant columns
268 data['Time'] = pd.to_numeric(data['Time'], errors='coerce')
269 data['Extension'] = pd.to_numeric(data['Extension'], errors='coerce')
270 data['Load'] = pd.to_numeric(data['Load'], errors='coerce')
271
272 # Drop rows with NaN values in essential columns
273 data = data.dropna(subset=['Time', 'Extension', 'Load'])
274
275 # Extract the columns
276 time = data['Time']
277 extension = data['Extension']
278 load = data['Load']
279
280 # Identifying the slack region (first non-zero significant load)
281 slack_threshold = 0.1 # N, threshold for detecting significant load
282 slack_index = (load > slack_threshold).idxmax() # Get index of first
283   significant load
284
285 # Slice the data from slack_index onwards
286 time_corrected = time[slack_index:].reset_index(drop=True) # Time starting
287   from slack point
288 u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(
289   drop=True) # Corrected extension
290 load_corrected = load[slack_index:].reset_index(drop=True) # Corresponding
291   load values
292
293 # Shift time_corrected to start from 0
294 time_corrected_shifted = time_corrected - time_corrected.min()
295
296 # Plot compensated Force-Displacement (F-u) curve
297 compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,

```

```
    mode='lines', name='Compensated F-u')
294 fig_fu = go.Figure(data=[compensated_force_displacement])
295
296 # Apply styling to the F-u figure
297 style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated
298     Force-Displacement (F-u) Curve')
299
300 # Save the F-u figure as PDF
301 fig_fu.write_image("fig-1-v1.pdf")
302
303 # Plot Force-Time (F-t) curve using corrected time
304 force_time = go.Scatter(x=time_corrected_shifted, y=load_corrected, mode='lines',
305     , name='F-t')
306 fig_ft = go.Figure(data=[force_time])
307
308 # Apply styling to the F-t figure
309 style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
310
311 # Save the F-t figure as PDF
312 fig_ft.write_image("fig-2-v1.pdf")
313
314 # Show both plots
315 fig_fu.show()
316 fig_ft.show()
317
318
319
320 L_0_rel = 69.17
321 L_0_rel
322
323
324 # In [180]:
325
326
327 # Identifying the slack region (first non-zero significant load)
328 slack_threshold = 0.1 # N, threshold for detecting when the load becomes
329     significant
330 slack_index = (load > slack_threshold).idxmax() # Get index of first
331     significant load
332 # Slice the data from slack_index onwards
333 time_corrected = time[slack_index: ].reset_index(drop=True) # Time starting
334     from slack point
335 u_corrected = (extension - extension[slack_index])[slack_index: ].reset_index(
336     drop=True) # Corrected extension
337 load_corrected = load[slack_index: ].reset_index(drop=True) # Corresponding
338     load values
339
340 # Plot compensated Force-Displacement (F-u) curve
341 compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
342     mode='lines', name='Compensated F-u')
343 layout_fu = go.Layout(
344     title='Compensated Force-Displacement (F-u) Curve',
```

```

339     xaxis=dict(title='Corrected Displacement u (mm)'),
340     yaxis=dict(title='Force F (N)'),
341 )
342 fig_fu = go.Figure(data=[compensated_force_displacement], layout=layout_fu)
343
344 # Plot Force-Time (F-t) curve using corrected time
345 force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name=
346   'F-t')
347 layout_ft = go.Layout(
348   title='Force-Time (F-t) Curve',
349   xaxis=dict(title='Time (s)'),
350   yaxis=dict(title='Force F (N)'),
351 )
352 fig_ft = go.Figure(data=[force_time], layout=layout_ft)
353
354 # Apply the styling to the F-u and F-t figures
355 style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated
356   Force-Displacement (F-u) Curve')
357 style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
358
359 # Show both plots
360 fig_fu.show()
361 fig_ft.show()
362
363
364 # In [170]:
365
366
367 extension = extension[slack_index:] - extension[slack_index]
368 load = load[slack_index:] - load[slack_index]
369
370
371 # In [152]:
372
373
374 # Define cross-sectional area (A) in mm^2
375 A = a * b # mm^2
376
377 # Calculate Engineering Stress (P)
378 stress = -load / A # Engineering stress in N/mm^2 or MPa
379
380 # Calculate Stretch (lambda) using the updated formula
381 stretch = 1 - (extension) / L_0_rel # Stretch ratio
382
383 # Calculate Engineering Strain (eps) as stretch - 1
384 strain = stretch - 1 # Engineering strain
385
386 # Plot Engineering Stress - Engineering Strain (P - eps) curve
387 fig_ps = go.Figure(data=[go.Scatter(x=strain, y=stress, mode='lines', name=
388   'Engineering Stress - Strain')])
389 style_plot(fig_ps, 'Engineering Strain (eps) [-]', 'Engineering Stress (P) [MPa
390   ]', 'Engineering Stress - Engineering Strain (P - eps) Curve')

```

```

389
390 # Plot Engineering Stress - Stretch (P - lambda) curve
391 fig_pl = go.Figure(data=[go.Scatter(x=stretch, y=stress, mode='lines', name='
392     Engineering Stress - Stretch')])  

393 style_plot(fig_pl, 'Stretch (lambda) [-]', 'Engineering Stress (P) [MPa]', 'Engineering
394     Stress - Stretch (P - lambda) Curve')
395
396 # Plot Engineering Stress - Time (P - t) curve
397 fig_pt = go.Figure(data=[go.Scatter(x=time_corrected, y=stress[slack_index:].
398     reset_index(drop=True), mode='lines', name='Engineering Stress - Time')])
399 style_plot(fig_pt, 'Time [s]', 'Engineering Stress (P) [MPa]', 'Engineering
400     Stress - Time (P - t) Curve')
401
402 # Show all plots
403 fig_ps.update_xaxes(range=[-0.8, 0])
404 fig_pl.update_xaxes(range=[0.2, 1])
405 fig_pt.update_xaxes(range=[0, 55])
406 fig_pt.update_yaxes(range=[-0.015, 0])
407
408 fig_ps.write_image("task2-1.pdf")
409 fig_pl.write_image("task2-2.pdf")
410 fig_pt.write_image("task2-3.pdf")
411
412
413 # TASK A - TASK 3
414
415 # Video
416
417 # In[11]:
418
419
420 import cv2
421
422 # Load the video
423 input_video_path = 'Relax_video_V2.mp4'
424 output_video_path = 'Relax_video_V2_binary.mp4'
425 cap = cv2.VideoCapture(input_video_path)
426
427 # Get video properties
428 fps = cap.get(cv2.CAP_PROP_FPS)
429 frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
430 frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
431 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
432
433 # Define the video writer for the output binary video
434 out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width,
435     frame_height), isColor=False)
436
437 # Process each frame
438 while cap.isOpened():

```

```
438     ret, frame = cap.read()
439     if not ret:
440         break
441
442     # Convert to grayscale
443     gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
444
445     # Apply binary threshold
446     _, binary_frame = cv2.threshold(gray_frame, 100, 255, cv2.THRESH_BINARY)
447
448     # Write the binary frame to the output video
449     out.write(binary_frame)
450
451 # Release video objects
452 cap.release()
453 out.release()
454
455 print(f"Binary video saved as '{output_video_path}'")
456
457
458 # In[57]:
459
460
461 import cv2
462 import numpy as np
463 import plotly.graph_objs as go
464
465 # Load the binary video
466 binary_video_path = 'Relax_video_V2_binary.mp4'
467 cap = cv2.VideoCapture(binary_video_path)
468
469 # Get video properties
470 fps = cap.get(cv2.CAP_PROP_FPS)
471 frames_per_half_second = int(fps / 2)
472 time_intervals = [] # Store the time intervals in seconds
473 white_pixel_sums = [] # Store the sum of white pixels for each half-second
474
475 current_sum = 0
476 frame_count = 0
477 time_point = 0.5 # Start with the first half-second interval
478
479 # Process each frame
480 while cap.isOpened():
481     ret, frame = cap.read()
482     if not ret:
483         break
484
485     # Count white pixels in the frame (255 is white in binary)
486     white_pixel_count = np.sum(frame == 255)
487     current_sum += white_pixel_count
488     frame_count += 1
489
490     # If half-second interval reached
491     if frame_count >= frames_per_half_second:
```

```
492     time_intervals.append(time_point)
493     white_pixel_sums.append(current_sum)
494
495     # Reset for the next half-second interval
496     current_sum = 0
497     frame_count = 0
498     time_point += 0.5
499
500 # Release the video capture object
501 cap.release()
502
503 # Remove the first data point from both lists
504 time_intervals = time_intervals[1:]
505 white_pixel_sums = white_pixel_sums[1:]
506
507 # Create the plot with Plotly
508 fig = go.Figure()
509 fig.add_trace(go.Scatter(x=time_intervals, y=white_pixel_sums, mode='lines+'
510                           markers, marker=dict(color='blue')))
511
512 # Update plot layout
513 fig.update_layout(
514     title='Sum of White Pixels in Each Half-Second Interval (First Point
515           Excluded)', xaxis_title='Time (s)', yaxis_title='Sum of White Pixels',
516     template="plotly_white")
517 )
518
519 fig.show()
520
521
522 # In [64]:
523
524
525 import plotly.graph_objs as go
526
527 # Assume 'white_pixel_sums' and 'time_intervals' are available from previous
528 # code
529 reference_sum = white_pixel_sums[1] # Using the 1-second value as the
# reference
530 stretch_ratios = [pixel_sum / reference_sum for pixel_sum in white_pixel_sums]
531
532 # Create the plot for the stretch ratio
533 fig = go.Figure()
534 fig.add_trace(go.Scatter(x=time_intervals, y=stretch_ratios, mode='lines+'
535                           markers, marker=dict(color='black')))
536
537 # Update plot layout
538 fig.update_layout(
539     title='Cross-Directional Stretch (lambda)', xaxis_title='Time (s)',
540     yaxis_title='Stretch (lambda)',
```

```
541     template="plotly_white"
542 )
543
544 style_plot(fig, 'Time [s]', 'Stretch (lambdaT) [-]', 'Cross-Directional Stretch
545             (lambda)')
546 fig.write_image("task3 -2.pdf")
547
548 fig.show()
549
550
551 # In[65]:
552
553
554 import plotly.graph_objs as go
555
556 # Calculate vertical stretch based on u_corrected
557 vertical_stretch = [(L_0 - u) / L_0 for u in u_corrected]
558
559 # Plot with vertical stretch on x-axis and cross-directional stretch (
560     # stretch_ratios) on y-axis
561 fig = go.Figure()
562
563 fig.add_trace(go.Scatter(
564     x=vertical_stretch,
565     y=stretch_ratios,
566     mode='lines+markers',
567     marker=dict(color='blue')
568 ))
569
570 # Update plot layout
571 fig.update_layout(
572     title='Cross-Directional Stretch vs Vertical Stretch',
573     xaxis_title='Stretch (lambda) [-]',
574     yaxis_title='Cross-Directional Stretch (lambdaT) [-]',
575     template="plotly_white"
576 )
577
578 style_plot(fig, 'Stretch (lambda) [-]', 'Cross-Directional Stretch (lambdaT)
579             [-]', 'Cross-Directional Stretch vs Vertical Stretch')
580 fig.write_image("task3 -3.pdf")
581
582 fig.show()
583
584
585 # In[68]:
586
587
588 import numpy as np
589 import plotly.graph_objs as go
590
591 # Step 1: Calculate vertical stretch based on u_corrected
592 vertical_stretch = [(L_0 - u) / L_0 for u in u_corrected]
```

```
592 # Step 2: Resample vertical stretch to match the length of stretch_ratios
593 num_vertical_stretch = len(vertical_stretch)
594 num_stretch_ratios = len(stretch_ratios)
595
596 # Generate original indices and new indices for resampling
597 original_indices = np.linspace(0, num_vertical_stretch - 1,
598     num_vertical_stretch)
599 new_indices = np.linspace(0, num_vertical_stretch - 1, num_stretch_ratios)
600
601 # Interpolate vertical stretch to match the size of stretch_ratios
602 vertical_stretch_resampled = np.interp(new_indices, original_indices,
603     vertical_stretch)
604
605 # Ensure the first and last values are the same
606 vertical_stretch_resampled[0] = vertical_stretch[0] # Keep the first value
607 vertical_stretch_resampled[-1] = vertical_stretch[-1] # Keep the last value
608
609 # Step 3: Plot with vertical stretch on x-axis and cross-directional stretch on
610 #          y-axis
611 fig = go.Figure()
612
613 fig.add_trace(go.Scatter(
614     x=vertical_stretch_resampled,
615     y=stretch_ratios,
616     mode='lines+markers',
617     marker=dict(color='blue')
618 ))
619
620 # Update plot layout
621 fig.update_layout(
622     title='Cross-Directional Stretch vs Vertical Stretch',
623     xaxis_title='Stretch (lambda) [-]',
624     yaxis_title='Cross-Directional Stretch (lambdaT) [-]',
625     template="plotly_white"
626 )
627
628 # Apply your styling function
629 style_plot(fig, 'Stretch (lambda) [-]', 'Cross-Directional Stretch (lambdaT) [-]', 'Cross-Directional Stretch vs Vertical Stretch')
630
631 # Save the plot as PDF
632 fig.write_image("task3-3.pdf")
633
634
635 # TASK A - TASK 4
636
637 # In [69]:
638
639
640 lambda_T_max = max(stretch_ratios)
641 lambda_T_max
```

```

642
643
644 # TASK A - TASK 5
645
646 # In[179]:
647
648
649 import numpy as np
650 import plotly.graph_objects as go
651
652 # Define initial cross-sectional area (A) in mm^2
653 A = a * b # initial cross-sectional area in mm^2
654
655 # Interpolate stretch_ratios and vertical_stretch to match the length of load
656 stretch_ratios_expanded = np.interp(np.arange(len(load)), np.linspace(0, len(
657     load)-1, len(stretch_ratios)), stretch_ratios)
658
659 # Calculate Stretch (lambda) as 1 - (extension / L_0_rel)
660 stretch = 1 - (extension) / L_0_rel # Expanded Stretch ratio
661
662 # Calculate True Stress (sig_true) as sig / (lambda^2)
663 true_stress = -load / (stretch_ratios_expanded ** 2 * A) # True stress using
664     actual cross-sectional area
665
666 # Calculate True Strain (eps_true) as ln(lambda)
667 true_strain = np.log(stretch) # True strain as natural log of stretch
668
669 # Plot True Stress - True Strain (sig_true - eps_true) curve
670 fig_tt = go.Figure(data=[go.Scatter(x=true_strain, y=true_stress, mode='lines',
671     name='True Stress - True Strain')])
672 style_plot(fig_tt, 'True Strain (eps_true) [-]', 'True Stress (sig_true) [MPa]',
673     'True Stress - True Strain (sig_true - eps_true) Curve')
674
675 # Set axis ranges (adjust as needed)
676 fig_tt.update_xaxes(range=[-1.4, 0.1])
677
678 # Save and display the plot
679 fig_tt.write_image("task2-true_stress_true_strain.pdf")
680 fig_tt.show()
681
682
683
684 def style_plot(fig, x_title, y_title, title):
685     # Add gradient background using shapes
686     fig.update_layout(
687         title=title,
688         plot_bgcolor='rgba(255, 255, 255, 0)', # Transparent plot background
689         shapes=[
690             dict(
691                 type='rect',

```

```

692         xref='paper', yref='paper',
693         x0=0, x1=1, y0=0, y1=1,
694         fillcolor='rgba(211, 211, 211, 0.15)', # Grey with 15% opacity
695         line=dict(width=0)
696     ),
697 ],
698 xaxis=dict(
699     title=x_title,
700     gridcolor='lightgrey', # Lighter grid color for more hollow effect
701     gridwidth=1,
702     zerolinecolor='black', # Zero line color
703     zerolinewidth=2,
704     showgrid=True,
705     linecolor='black' # X-axis line color
706 ),
707 yaxis=dict(
708     title=y_title,
709     gridcolor='lightgrey', # Lighter grid color for more hollow effect
710     gridwidth=1,
711     zerolinecolor='black', # Zero line color
712     zerolinewidth=2,
713     showgrid=True,
714     linecolor='black' # Y-axis line color
715 ),
716 font=dict(color='black'),
717 )
718
719 # Set colors for traces
720 if len(fig.data) >= 1:
721     fig.update_traces(line=dict(color='red'), selector=dict(name='
Compensated F-u')) # Default color for the first trace
722 if len(fig.data) >= 2:
723     fig.data[1].line.color = 'green' # Set the second trace color to green
724
725
726 # In [97]:
727
728
729 # Import necessary libraries
730 import pandas as pd
731 import plotly.graph_objs as go
732
733 # Define basic parameters
734 H_0_cyc = 75 # mm, initial tool separation
735 u_1 = 10.5 # mm, displacement of the first step in the uploading
736 delta_u = 5.5 # mm, incremental displacement
737 N = 9 # Number of increments
738 v_cyc = 40 # mm/min, cross-head speed during cyclic test
739 delta_t_rel = 30 # s, relaxation time
740 a_cyc = 77.71 # mm
741 b_cyc = 75.11 # mm
742 A_cyc = a_cyc * b_cyc
743
744 # Load the cyclic test CSV file into a DataFrame

```

```

745 file_name_cyc = 'CYCLIC_RawData_18.csv'
746 data_cyc = pd.read_csv(file_name_cyc)
747
748 # Convert columns to appropriate data types
749 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce') # Convert Time to float
750 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce') # Convert Extension to float
751 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce') # Convert Load to float
752
753 # Extract the columns: Time, Extension, Load
754 time_cyc = data_cyc['Time']
755 extension_cyc = data_cyc['Extension']
756 load_cyc = data_cyc['Load']
757
758 # Define the exact height of the cyclic test specimen (L_0^(cyc))
759 L_0_cyc = H_0_cyc - u_1 # mm
760
761 # Perform slack correction for displacement (extension)
762 slack_threshold_cyc = 0.1 # N, threshold for detecting when the load becomes significant
763 slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax() # Get index of first significant load
764
765 # Slice the data from slack_index onwards
766 time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True) # Time starting from slack point
767 u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc])[slack_index_cyc:].reset_index(drop=True) # Corrected extension
768 load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True) # Corresponding load values
769
770 # Plot compensated Force-Displacement (F-u) curve
771 compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=load_corrected_cyc, mode='lines', name='Compensated F-u')
772 fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc])
773 style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated Force-Displacement (F-u) Curve')
774
775 # Plot Force-Time (F-t) curve using corrected time
776 force_time_cyc = go.Scatter(x=time_corrected_cyc, y=load_corrected_cyc, mode='lines', name='F-t')
777 fig_ft_cyc = go.Figure(data=[force_time_cyc])
778 style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
779
780 fig_fu_cyc.write_image("taskB1-1.pdf")
781
782 fig_ft_cyc.write_image("taskB1-2.pdf")
783
784 # Show both plots
785 fig_fu_cyc.show()
786 fig_ft_cyc.show()
787

```

```

788
789 # In[95]:
790
791
792 # Import necessary libraries
793 import pandas as pd
794 import plotly.graph_objs as go
795
796 # Define basic parameters
797 H_0_cyc = 75 # mm, initial tool separation
798 u_1 = 10.5 # mm, displacement of the first step in the uploading
799 delta_u = 5.5 # mm, incremental displacement
800 N = 9 # Number of increments
801 v_cyc = 40 # mm/min, cross-head speed during cyclic test
802 delta_t_rel = 30 # s, relaxation time
803
804 # Load the cyclic test CSV file into a DataFrame
805 file_name_cyc = 'CYCLIC_RawData_18.csv'
806 data_cyc = pd.read_csv(file_name_cyc)
807
808 # Convert columns to appropriate data types
809 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce') # Convert Time to float
810 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce') # Convert Extension to float
811 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce') # Convert Load to float
812
813 # Extract the columns: Time, Extension, Load
814 time_cyc = data_cyc['Time']
815 extension_cyc = data_cyc['Extension']
816 load_cyc = data_cyc['Load']
817
818 # Define the exact height of the cyclic test specimen (L_0^(cyc))
819 L_0_cyc = H_0_cyc - 6.3 # mm
820
821 # Plot Force-Displacement (F-u) curve without slack correction
822 force_displacement_cyc = go.Scatter(x=extension_cyc, y=load_cyc, mode='lines', name='F-u')
823 fig_fu_cyc = go.Figure(data=[force_displacement_cyc])
824 style_plot(fig_fu_cyc, 'Displacement u (mm)', 'Force F (N)', 'Force-Displacement (F-u) Curve')
825
826 # Plot Force-Time (F-t) curve without slack correction
827 force_time_cyc = go.Scatter(x=time_cyc, y=load_cyc, mode='lines', name='F-t')
828 fig_ft_cyc = go.Figure(data=[force_time_cyc])
829 style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
830
831 # Save plots as PDFs
832 fig_fu_cyc.write_image("taskB1-1-noslack.pdf")
833 fig_ft_cyc.write_image("taskB1-2-noslack.pdf")
834
835 # Show both plots
836 fig_fu_cyc.show()

```

```

837 fig_ft_cyc.show()
838
839
840 # In[96]:
841
842
843 # Import necessary libraries
844 import pandas as pd
845 import plotly.graph_objs as go
846
847 # Define basic parameters
848 H_0_cyc = 75 # mm, initial tool separation
849 u_1 = 10.5 # mm, displacement of the first step in the uploading
850 delta_u = 5.5 # mm, incremental displacement
851 N = 9 # Number of increments
852 v_cyc = 40 # mm/min, cross-head speed during cyclic test
853 delta_t_rel = 30 # s, relaxation time
854
855 # Load the cyclic test CSV file into a DataFrame
856 file_name_cyc = 'CYCLIC_RawData_18.csv',
857 data_cyc = pd.read_csv(file_name_cyc)
858
859 # Convert columns to appropriate data types
860 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce') # Convert Time to float
861 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce') # Convert Extension to float
862 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce') # Convert Load to float
863
864 # Extract the columns: Time, Extension, Load
865 time_cyc = data_cyc['Time']
866 extension_cyc = data_cyc['Extension']
867 load_cyc = data_cyc['Load']
868
869 # Define the exact height of the cyclic test specimen (L_0^(cyc))
870 L_0_cyc = H_0_cyc - 6.3 # mm
871
872 # Plot Force-Displacement (F-u) curve without slack correction
873 force_displacement_cyc = go.Scatter(x=extension_cyc, y=load_cyc, mode='lines', name='F-u')
874 fig_fu_cyc = go.Figure(data=[force_displacement_cyc])
875 style_plot(fig_fu_cyc, 'Displacement u (mm)', 'Force F (N)', 'Force-Displacement (F-u) Curve')
876
877 # Plot Force-Time (F-t) curve without slack correction
878 force_time_cyc = go.Scatter(x=time_cyc, y=load_cyc, mode='lines', name='F-t')
879 fig_ft_cyc = go.Figure(data=[force_time_cyc])
880 style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
881
882 fig_fu_cyc.update_xaxes(range=[5, 8]) # Set x-axis range
883 fig_fu_cyc.update_yaxes(range=[-2, 7])
884
885 # Save plots as PDFs

```

```
886 fig_fu_cyc.write_image("taskB1-1-noslack-close.pdf")
887 fig_ft_cyc.write_image("taskB1-2-noslack.pdf")
888
889 # Show both plots
890 fig_fu_cyc.show()
891 fig_ft_cyc.show()
892
893
894 # TASK B - TASK 2
895
896 # In[99]:
897
898
899 extension_cyc = extension_cyc[slack_index_cyc:] - extension_cyc[slack_index_cyc]
900 load_cyc = load_cyc[slack_index_cyc:] - load_cyc[slack_index_cyc]
901
902
903 # In[104]:
904
905
906 L_0_cyc = H_0_cyc - 6.5
907
908
909 # In[125]:
910
911
912 # Calculate engineering stress (P)
913 engineering_stress = load_cyc / A_cyc *-1
914 stretch = 1 - extension_cyc/L_0_cyc
915 engineering_strain = stretch - 1 # Extension divided by initial length
916
917 # Plot Engineering Stress vs Engineering Strain (P - eps)
918 fig_stress_strain = go.Figure()
919 fig_stress_strain.add_trace(go.Scatter(x=engineering_strain, y=
920     engineering_stress, mode='lines', name='P - eps'))
921 style_plot(fig_stress_strain, 'Engineering Strain (eps)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Engineering Strain (P - eps)')
922 fig_stress_strain.update_xaxes(range=[-0.8, 0.1]) # Set x-axis range
923
924 fig_stress_strain.write_image("taskB2-1.pdf")
925 fig_stress_strain.show()
926
927 # Plot Engineering Stress vs Stretch (P - lambda)
928 fig_stress_stretch = go.Figure()
929 fig_stress_stretch.add_trace(go.Scatter(x=stretch, y=engineering_stress, mode='
930     lines', name='P - lambda'))
931 style_plot(fig_stress_stretch, 'Stretch (lambda)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Stretch (P - lambda)')
932 fig_stress_stretch.update_xaxes(range=[0.2, 1.1]) # Set x-axis range
933
934 fig_stress_stretch.write_image("taskB2-2.pdf")
935 fig_stress_stretch.show()
```

```

935 # Plot Engineering Stress vs Time (P - t)
936 fig_stress_time = go.Figure()
937 fig_stress_time.add_trace(go.Scatter(x=time_cyc, y=engineering_stress, mode='lines', name='P - t'))
938 style_plot(fig_stress_time, 'Time (s)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Time (P - t)')
939
940 fig_stress_time.write_image("taskB2-3.pdf")
941 fig_stress_time.show()
942
943
944 # TASK B - TASK 3
945
946 # In[126]:
947
948
949 import pandas as pd
950 import numpy as np
951 import plotly.graph_objs as go
952 from scipy.signal import argrelextrema
953
954 # Load the cyclic test CSV file into a DataFrame
955 file_name_cyc = 'CYCLIC_RawData_18.csv',
956 data_cyc = pd.read_csv(file_name_cyc)
957
958 # Convert columns to appropriate data types
959 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')
960 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')
961 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')
962
963 # Extract columns: Time, Extension, Load
964 time_cyc = data_cyc['Time']
965 extension_cyc = data_cyc['Extension']
966 load_cyc = data_cyc['Load']
967
968 # Define parameters
969 L_0_cyc = 68.5 # mm, original height of the specimen
970
971 # Perform slack correction for displacement (extension)
972 slack_threshold_cyc = 0.1 # N, threshold for detecting when load becomes significant
973 slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax() # Index of first significant load
974
975 # Slice the data from slack_index onwards
976 time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True)
977 u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc])[slack_index_cyc:].reset_index(drop=True)
978 load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True)
979
980 # Smoothing load data to reduce noise for extrema detection
981 smoothed_load = load_corrected_cyc.rolling(window=300, center=True).mean() # Adjust as necessary
982

```

```

983 # Plotting for visualization of engineering stress vs stretch
984 stress_vs_stretch = go.Scatter(x=stretch, y=engineering_stress, mode='lines',
985     name='Engineering Stress vs Stretch')
986 fig_stress_stretch = go.Figure(data=[stress_vs_stretch])
987 fig_stress_stretch.update_layout(title='Engineering Stress vs Stretch',
988                                 xaxis_title='Stretch (lambda)',
989                                 yaxis_title='Engineering Stress (MPa)')
990 fig_stress_stretch.show()
991
992 # Print the lists of unloading maxima and uploading minima
993 print("Unloading Maxima:", unloading_max_values)
994 print("Uploading Minima:", uploading_min_values)
995
996 # In[181]:
997
998
999 import pandas as pd
1000 import numpy as np
1001 import plotly.graph_objs as go
1002 from scipy.signal import argrelextrema
1003
1004 # Load the cyclic test CSV file into a DataFrame
1005 file_name_cyc = 'CYCLIC_RawData_18.csv'
1006 data_cyc = pd.read_csv(file_name_cyc)
1007
1008 # Convert columns to appropriate data types
1009 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')
1010 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')
1011 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')
1012
1013 # Extract columns: Time, Extension, Load
1014 time_cyc = data_cyc['Time']
1015 extension_cyc = data_cyc['Extension']
1016 load_cyc = data_cyc['Load']
1017
1018 # Define the exact height of the cyclic test specimen (L_0^(cyc))
1019 L_0_cyc = 68.5 # mm
1020
1021 # Perform slack correction for displacement (extension)
1022 slack_threshold_cyc = 0.1 # N, threshold for detecting when load becomes
1023     significant
1024 slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax() # Index of first
1025     significant load
1026
1027 # Slice the data from slack_index onwards
1028 time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True)
1029 u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc:])[slack_index_cyc:].
1030     reset_index(drop=True)
1031 load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True)
1032
1033 # Smoothing load data to reduce noise for extrema detection
1034 smoothed_load = load_corrected_cyc.rolling(window=300, center=True).mean() # Adjust as necessary

```

```

1032
1033 # Find the global maximum index separating uploading and unloading phases
1034 global_max_index = smoothed_load.idxmax()
1035
1036 # Find approximate local maxima in the unloading phase (after the global
1037 # maximum)
1038 unloading_approx_max_indices = argrelextrema(smoothed_load[global_max_index:].
1039     values, np.greater, order=70)[0] + global_max_index
1040
1041 # Refine local maxima by ensuring only one maximum per cycle
1042 unloading_max_indices = []
1043 for i in range(1, len(unloading_approx_max_indices)):
1044     start_idx = unloading_approx_max_indices[i-1]
1045     end_idx = unloading_approx_max_indices[i]
1046     exact_max_idx = load_corrected_cyc[start_idx:end_idx].idxmax()
1047     unloading_max_indices.append(exact_max_idx)
1048
1049 # Extract values of the exact unloading maxima
1050 unloading_max_values = load_corrected_cyc[unloading_max_indices].values
1051
1052 # Find approximate local minima in the uploading phase (before the global
1053 # maximum)
1054 uploading_approx_min_indices = argrelextrema(smoothed_load[:global_max_index].
1055     values, np.less, order=50)[0]
1056
1057 # Add the last index to ensure we include it in the search for the final local
1058 # minimum
1059 if global_max_index < len(smoothed_load):
1060     uploading_approx_min_indices = np.append(uploading_approx_min_indices,
1061         global_max_index)
1062
1063 # Refine local minima by ensuring only one minimum per cycle
1064 uploading_min_indices = []
1065 for i in range(1, len(uploading_approx_min_indices)):
1066     start_idx = uploading_approx_min_indices[i-1]
1067     end_idx = uploading_approx_min_indices[i]
1068     exact_min_idx = load_corrected_cyc[start_idx:end_idx].idxmin()
1069     uploading_min_indices.append(exact_min_idx)
1070
1071 # Check the last segment from the last detected minimum to the global maximum
1072 if uploading_min_indices:
1073     last_segment_min_idx = load_corrected_cyc[uploading_min_indices[-1]:.
1074         global_max_index].idxmin()
1075     if last_segment_min_idx not in uploading_min_indices:
1076         uploading_min_indices.append(last_segment_min_idx)
1077
1078 # Extract values of the exact uploading minima
1079 uploading_min_values = load_corrected_cyc[uploading_min_indices].values
1080
1081 # Plotting for visualization
1082 compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=
1083     load_corrected_cyc, mode='lines', name='Compensated F-u')
1084 unloading_max_scatter = go.Scatter(x=u_corrected_cyc[unloading_max_indices], y=
1085     unloading_max_values, mode='markers', name='Unloading Local Maxima', marker=

```

```

    dict(color='red', size=8))
1077 uploading_min_scatter = go.Scatter(x=u_corrected_cyc[uploading_min_indices], y=
1078     uploading_min_values, mode='markers', name='Uploading Local Minima', marker=
1079     dict(color='blue', size=8))

1080 fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc,
1081     unloading_max_scatter, uploading_min_scatter])
1082 style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', ,
1083     'Compensated Force-Displacement (F-u) Curve with Local Maxima and Minima')
1084 fig_fu_cyc.show()

1085 # Print the lists of unloading maxima and uploading minima
1086 print("Unloading Maxima:", unloading_max_values)
1087 print("Uploading Minima:", uploading_min_values)

1088 # In [182]:
1089
1090
1091 unloading_max_values = unloading_max_values[:-2]
1092
1093 # Print the updated unloading maxima list
1094 print("Updated Unloading Maxima:", unloading_max_values)
1095 print("Uploading Minima:", uploading_min_values)

1096
1097
1098 # In [183]:
1099
1100
1101 # Plotting for visualization
1102 compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=
1103     load_corrected_cyc, mode='lines', name='Compensated F-u')
1104 unloading_max_scatter = go.Scatter(x=u_corrected_cyc[unloading_max_indices], y=
1105     unloading_max_values, mode='markers', name='Unloading Local Maxima', marker=
1106     dict(color='red', size=8))
1107 uploading_min_scatter = go.Scatter(x=u_corrected_cyc[uploading_min_indices], y=
1108     uploading_min_values, mode='markers', name='Uploading Local Minima', marker=
1109     dict(color='blue', size=8))

1110 fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc,
1111     unloading_max_scatter, uploading_min_scatter])
1112 style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', ,
1113     'Compensated Force-Displacement (F-u) Curve with Local Maxima and Minima')
1114 fig_fu_cyc.show()

1115 # Assuming unloading_max_values and uploading_min_values are already defined
1116 # and have the same length
1117 midpoints = [(unloading_max_values[i] + uploading_min_values[i]) / 2 for i in
1118     range(len(unloading_max_values))]
```

```

1117 # Print the midpoints list
1118 print("Midpoints:", midpoints)
1119
1120
1121 # In [185]:
1122
1123
1124 # Assuming you have the following lists:
1125 # unloading_max_values, uploading_min_values
1126 # And you also have the corresponding extension values in 'u_corrected_cyc'
1127
1128 # Ensure the lists have the same length
1129 if len(unloading_max_values) == len(uploading_min_values):
1130     midpoints = []
1131     midpoint_x_values = []
1132
1133     for i in range(len(unloading_max_values)):
1134         midpoint_value = (unloading_max_values[len(unloading_max_values)-i-1] +
1135                             uploading_min_values[i]) / 2
1136         midpoint_x_value = u_corrected_cyc[uploading_min_indices[i]] # Use the
1137             corresponding index for x value
1138
1139         midpoints.append(midpoint_value)
1140         midpoint_x_values.append(midpoint_x_value)
1141
1142     # Print the midpoints and their corresponding x-values
1143     for x, midpoint in zip(midpoint_x_values, midpoints):
1144         print(f"X: {x}, Midpoint: {midpoint}")
1145 else:
1146     print("Error: unloading_max_values and uploading_min_values must have the
1147 same length.")
1148
1149
1150 # Plotting for visualization
1151 compensated_force_displacement_cyc = go.Scatter(
1152     x=u_corrected_cyc,
1153     y=load_corrected_cyc,
1154     mode='lines',
1155     name='Compensated F-u',
1156 )
1157
1158 unloading_max_scatter = go.Scatter(
1159     x=u_corrected_cyc[unloading_max_indices],
1160     y=unloading_max_values,
1161     mode='markers',
1162     name='Unloading Local Maxima',
1163     marker=dict(color='red', size=8)
1164 )
1165
1166 uploading_min_scatter = go.Scatter(
1167     x=u_corrected_cyc[uploading_min_indices],

```

```

1168     y=uploading_min_values,
1169     mode='markers',
1170     name='Uploading Local Minima',
1171     marker=dict(color='blue', size=8)
1172 )
1173
1174 # Create a scatter plot for midpoints
1175 midpoints_scatter = go.Scatter(
1176     x=midpoint_x_values, # Corresponding x-values for midpoints
1177     y=midpoints,          # Midpoint values
1178     mode='markers',
1179     name='Midpoints',
1180     marker=dict(color='green', size=8, symbol='diamond')
1181 )
1182
1183 # Create the figure with all traces
1184 fig_fu_cyc = go.Figure(data=[
1185     compensated_force_displacement_cyc,
1186     unloading_max_scatter,
1187     uploading_min_scatter,
1188     midpoints_scatter # Add midpoints to the figure
1189 ])
1190
1191 # Style and show the plot
1192 style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', '
1193     Compensated Force-Displacement (F-u) Curve with Local Maxima, Minima, and
1194     Midpoints')
1195 fig_fu_cyc.show()
1196
1197
1198
1199 # In [189]:
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219

```

```

1220     name='Compensated F-u'
1221 )
1222
1223 unloading_max_scatter = go.Scatter(
1224     x=u_corrected_cyc[unloading_max_indices],
1225     y=unloading_max_values,
1226     mode='markers',
1227     name='Unloading Local Maxima',
1228     marker=dict(color='red', size=8)
1229 )
1230
1231 uploading_min_scatter = go.Scatter(
1232     x=u_corrected_cyc[uploading_min_indices],
1233     y=uploading_min_values,
1234     mode='markers',
1235     name='Uploading Local Minima',
1236     marker=dict(color='blue', size=8)
1237 )
1238
1239 # Midpoints scatter plot
1240 midpoints_scatter = go.Scatter(
1241     x=midpoint_x_values,
1242     y=midpoints,
1243     mode='markers',
1244     name='Midpoints',
1245     marker=dict(color='green', size=8, symbol='diamond')
1246 )
1247
1248 # Polynomial line for midpoints
1249 polynomial_line = go.Scatter(
1250     x=x_poly,
1251     y=y_poly,
1252     mode='lines',
1253     name='Polynomial Fit',
1254     line=dict(color='black', width=2)
1255 )
1256
1257 # Create the figure with all traces
1258 fig_fu_cyc = go.Figure(data=[
1259     compensated_force_displacement_cyc,
1260     unloading_max_scatter,
1261     uploading_min_scatter,
1262     midpoints_scatter,
1263     polynomial_line # Add polynomial line to the figure
1264 ])
1265
1266 # Style and show the plot
1267 style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', ,
1268             'Compensated Force-Displacement (F-u) Curve with Local Maxima, Minima,
1269             Midpoints, and Polynomial Fit')
fig_fu_cyc.show()
1270
1271 # TASK B - TASK 4

```

```
1272
1273 # In [190]:
1274
1275
1276 from PIL import Image
1277 import numpy as np
1278
1279 # Load the images
1280 image_kicsi = Image.open('taskB4-2_kicsi.png').convert('L') # Convert to
   grayscale
1281 image_nagy = Image.open('taskB4-2_nagy.png').convert('L')      # Convert to
   grayscale
1282
1283 # Convert images to numpy arrays
1284 array_kicsi = np.array(image_kicsi)
1285 array_nagy = np.array(image_nagy)
1286
1287 # Count white pixels (assuming white is 255 in binary images)
1288 num_white_kicsi = np.sum(array_kicsi == 255)
1289 num_white_nagy = np.sum(array_nagy == 255)
1290
1291 # Calculate the ratio
1292 if num_white_nagy != 0:
1293     ratio = num_white_nagy / num_white_kicsi
1294 else:
1295     ratio = None # Handle case where there are no white pixels in the large
   image
1296
1297 # Print results
1298 print(f"Number of white pixels in taskB4-2_kicsi: {num_white_kicsi}")
1299 print(f"Number of white pixels in taskB4-2_nagy: {num_white_nagy}")
1300 print(f"Ratio (taskB4-2_nagy / taskB4-2_kicsi): {ratio}")
```